

**CTS-300 System
User's Guide**

**Order No. AA-C747E-TC
June 1983**

software **digital**

REVISION HISTORY

Effective with the release of the CTS-300 Version 8 software, this user's guide has been revised to reflect changes resulting from the software changes and to provide you with a more useful manual.

The highlights of the documentation changes that have been made for Version 8 are:

- The DIBOL compiler is now supplied in two forms; one for use in the SUD environment and one for use in the XMTSD environment. The DIBOL compiler is described in Chapter 4.
- There are some changes in the CTSGEN questions. These are documented in Chapter 6.
- The SORT/MERGE program name has been changed to SORTGEN to better describe its function. SORTGEN is described in Chapter 11.
- The DIBOL SEND and RECV statements now function in an SUD environment. Chapter 5 discusses the new message file used to implement this new capability. The STATUS utility now reports information on messages when operating under SUD. This function is described in Chapter 13.
- The background information on ISAM files has been moved to Appendix A, leaving Chapter 10 to cover ISMUTL (the ISAM utility) only.
- Appendix B contains examples of the link structure of all the CTS-300 modules. These examples are to be used as a guide when re-linking after making patches.
- The commands available in the time-sharing environment (TSD and XMTSD) have been moved to Appendix C for easier reference.
- The previous DIBOL language reference manual issued with CTS-300 has been replaced by the *DIBOL-83 Language Reference Manual*. This new manual describes the language as it is implemented on all DIGITAL operating systems. Differences in operation between systems is covered in another new manual entitled the *DIBOL-83 Compatability Guide*. DIBOL external subroutines that are available only on CTS-300 are described in Appendix D of this user's guide.
- Appendix E contains general informaton on data I/O with CTS-300.

CTS-300 System User's Guide

**Order No. AA-C747E-TC
June 1983**

SUPERSESSION/UPDATE INFORMATION:

This is the fourth revision of this manual.

OPERATING SYSTEM AND VERSION:

RT-11 V5

SOFTWARE VERSION:

CTS-300 V8

Fourth Revision, June 1983

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The specifications and drawings, herein, are the property of Digital Equipment Corporation and shall not be reproduced or copied or used in whole or in part as the basis for the manufacture or sale of items without written permission.

Copyright © 1983 by Digital Equipment Corporation. All Rights Reserved

The following are trademarks of Digital Equipment Corporation:


CTI BUS	MASSBUS	RSTS
DEC	PDP	RSX
DECmate	P/OS	Tool Kit
DECsystem-10	PRO/BASIC	UNIBUS
DECSYSTEM-20	Professional	VAX
DECUS	PRO/FMS	VMS
DECwriter	PRO/RMS	VT
DIBOL	PROSE	Work Processor
	Rainbow	

TABLE OF CONTENTS

	Page
SECTION I	OVERVIEW OF CTS-300
CHAPTER 1	INTRODUCTION TO CTS-300
1.1	INTRODUCTION 1-1
1.2	MONITORS 1-1
1.2.1	SJ Monitor 1-1
1.2.2	FB Monitor 1-1
1.2.3	XM Monitor 1-2
1.3	RUN-TIME SYSTEMS 1-2
1.3.1	The Single-User System 1-2
1.3.2	The Time-Shared System 1-2
1.3.3	The Extended Memory Time-Shared System 1-3
1.4	DEVELOPMENT 1-3
1.4.1	Program Development 1-3
1.4.2	System Development 1-3
1.5	DIBOL UTILITIES AND PROGRAMS 1-3
1.5.1	Supplied Utilities 1-3
1.5.2	Other DIBOL Programs 1-4
CHAPTER 2	BASIC COMMANDS AND FILE CONVENTIONS
2.1	INTRODUCTION 2-1
2.2	COMMANDS 2-1
2.2.1	Special Character Functions 2-1
2.2.2	Basic Command Rules 2-2
2.2.2.1	Commands to Allocate System Resources 2-2
2.2.2.2	Commands to Start a Program 2-3
2.2.3	Monitor Error Messages 2-4
2.3	FILE CONVENTIONS 2-4
2.3.1	File Naming 2-4
2.3.2	File Types 2-4
2.3.3	CTS-300 Data Files 2-5
2.3.3.1	Sequential and Random Access Sequential Files 2-5
2.3.3.2	Multivolume Sequential Files 2-5
2.3.3.3	ISAM Files 2-7
SECTION II	DEVELOPMENT
CHAPTER 3	DIBOL KEYBOARD EDITOR
3.1	INTRODUCTION 3-1
3.1.1	Requirements and Limitations 3-1
3.1.2	Characteristics 3-1
3.1.3	Chapter Organization 3-1

TABLE OF CONTENTS (Cont.)

		Page
3.2	OPERATION	3-2
3.2.1	Operating System Set Up	3-2
3.2.2	Terminology	3-2
3.2.3	Terminal Characteristics	3-2
3.2.4	Running DKED	3-2
3.2.5	Stopping DKED	3-3
3.2.6	Messages	3-3
3.2.7	The HELP Frame	3-3
3.3	CREATING AND HANDLING FILES	3-5
3.3.1	DKED File Characteristics	3-5
3.3.2	Creating New Files	3-7
3.3.3	Existing Files as Input	3-8
3.3.4	Specifying Output File Size	3-9
3.3.5	File Backup	3-10
3.3.6	File Manipulation Commands	3-10
3.3.6.1	EXIT Command	3-11
3.3.6.2	QUIT Command	3-11
3.3.6.3	REOPEN Command	3-11
3.4	CURSOR CONTROL	3-11
3.4.1	ADVANCE	3-13
3.4.2	BACKUP	3-13
3.4.3	TOP	3-13
3.4.4	BOTTOM	3-14
3.4.5	BEGIN OF LINE (Beginning of Line)	3-14
3.4.6	EOL (End of Line)	3-14
3.4.7	WORD	3-14
3.4.8	PAGE	3-14
3.4.9	SECTION	3-14
3.4.10	FIND (Search)	3-14
3.4.10.1	Mode	3-15
3.4.10.2	Model	3-15
3.4.10.3	Set Search BEGIN/END	3-15
3.4.10.4	Set Exact NONE/CASEFLAG	3-16
3.4.10.5	Set Wildcards ON/OFF	3-16
3.4.10.6	Search HELP Frames	3-17
3.4.10.7	Using FIND	3-18
3.4.10.8	Error Conditions	3-19
3.4.11	FIND NEXT (Continue Search)	3-19
3.4.12	Arrow Keys	3-19
3.4.13	RETURN Key	3-20
3.5	EDITING	3-20
3.5.1	DELCHR (Delete Character)	3-22
3.5.2	UNDLCH (Undelete Character)	3-22
3.5.3	DELIN (Delete Line)	3-22
3.5.4	UNDELIN (Undelete Line)	3-22
3.5.5	DELEOL (Delete End of Line)	3-22

TABLE OF CONTENTS (Cont.)

		Page
3.5.6	CHNGCASE (Change Case)	3-23
3.5.7	APPEND	3-23
3.5.8	REPLACE	3-23
3.5.9	SUBS (Substitute)	3-23
3.5.10	OPEN LINE	3-23
3.5.11	SELECT	3-23
3.5.12	RESET	3-24
3.5.13	CUT	3-24
3.5.14	PASTE	3-24
3.5.15	YANK	3-24
3.5.16	RUB CHAR OUT (Rub out)	3-24
CHAPTER	4	DIBOL COMPILER
	4.1	INTRODUCTION
	4.1.1	Characteristics
	4.1.2	Chapter Organization
	4.2	USING DICOMP
	4.2.1	Running DICOMP
	4.2.2	Command Qualifiers
	4.2.3	Specifying Output Files
	4.2.4	Contents of the Listing File
	4.2.4.1	The Program Listing
	4.2.4.2	The Symbol Table
	4.2.4.3	The Label Table
	4.2.3	Standard Listings
	4.2.4	CREF Listing
	4.2.4.1	Common Cross-Reference Table
	4.2.4.2	Label Cross-Reference Table
	4.2.4.3	Symbol Cross-Reference Table
	4.2.4.4	External Subroutine Cross-Reference Table
	4.3	ERROR MESSAGES
CHAPTER	5	OPERATING AND RUN-TIME SYSTEMS
	5.1	INTRODUCTION
	5.1.1	General Requirements
	5.1.2	Chapter Organization
	5.2	GENERAL SYSTEM CHARACTERISTICS
	5.2.1	Foreground/Background Operation
	5.2.2	Using Magnetic Tape
	5.2.3	Using the Error Logger
	5.2.4	Utilizing Resources on a Small System
	5.2.5	Stopping Programs
	5.2.6	CTS-300 System Function
	5.3	THE SINGLE-USER ENVIRONMENT
	5.3.1	System Requirements for SUD
	5.3.3	SEND/RECV for SUD

TABLE OF CONTENTS (Cont.)

		Page
5.3.3	Preparing Programs	5-7
5.3.4	Linking Programs	5-7
5.3.5	Running Programs	5-8
5.3.6	SUD Memory Allocation	5-8
5.4	THE TIME-SHARED ENVIRONMENT	5-9
5.4.1	System Requirements for Time-Sharing	5-9
5.4.2	Dynamic Memory Allocation	5-10
5.4.3	Scheduling	5-11
5.4.4	Detached Program Operation	5-11
5.4.5	Data File Management	5-11
5.4.5.1	File Sharing	5-11
5.4.5.2	Device Sharing	5-13
5.4.6	Preparing Programs	5-13
5.4.7	Linking	5-13
5.4.7.1	Linking to the Time-Shared DIBOL Library	5-14
5.4.7.2	Linking for DDT Use	5-14
5.4.8	Creating Overlays	5-14
5.4.9	Time-Shared Keyboard Commands	5-15
5.4.10	Programmed Startup	5-15
5.4.10.1	Forced Job Startup	5-15
5.4.10.2	Chain Mode Startup	5-16
5.4.10.3	Implicit Job Startup	5-16
5.5	TSD CHARACTERISTICS	5-17
5.5.1	System Requirements for TSD	5-17
5.5.2	Running the TSD Run-Time System Program	5-17
5.5.3	TSD Memory Allocation	5-18
5.6	XMTSD CHARACTERISTICS (Background Operation)	5-18
5.6.1	System Requirements	5-18
5.6.2	Running XMTSD in the Background	5-18
5.6.3	Memory Allocation	5-19
5.6.4	Applications	5-20
5.7	XMTSD CHARACTERISTICS (Foreground Operation)	5-20
5.7.1	System Requirements	5-20
5.7.2	Running XMTSD in the Foreground	5-21
5.7.3	Memory Allocation	5-22
5.7.4	Foreground Queue Program	5-23
5.7.5	Background Listener Program	5-24
5.7.6	Foreground/Background Communications Commands	5-24
5.7.6.1	The SUBMIT Communication Command	5-25
5.7.6.2	The SHOW Communication Command	5-26
5.7.6.3	The CANCEL Communication Command	5-27
5.7.7	Applications	5-27
5.8	SUMMARY OF XMTSD CAPABILITIES	5-28
5.9	TERMINATING TIME-SHARED OPERATION (RTEXT)	5-29
5.9.1	Running RTEXT	5-29
5.9.2	Chaining to RTEXT	5-29
5.9.3	RTEXT for XMTSD in the Foreground	5-30

TABLE OF CONTENTS (Cont.)

	Page
CHAPTER 6	SYSTEM DEVELOPMENT
6.1	INTRODUCTION 6-1
6.1.1	System Generation Programs 6-1
6.1.2	Chapter Organization 6-1
6.1.3	CTSGEN Errors 6-1
6.2	SYSGEN 6-2
6.3	CTSGEN 6-3
6.3.1	Characteristics 6-3
6.3.1.1	Choices 6-3
6.3.1.2	Preliminary Requirements 6-4
6.3.1.3	Question Types 6-5
6.3.2	CTSGEN Dialog 6-5
6.3.2.1	Single-User or Time-Shared System 6-7
6.3.2.2	Single-User System 6-7
6.3.2.3	Time-Shared System 6-8
6.3.2.4	Terminal Specification 6-9
6.3.2.5	Local DL11 Terminals 6-9
6.3.2.6	Remote DL11 Terminals 6-10
6.3.2.7	Local DZ11 Terminals 6-12
6.3.2.8	Remote DZ11 Terminals 6-13
6.3.2.9	System Hardware/Software Configuration 6-14
6.3.2.10	Naming the Time-Shared Program 6-19
SECTION III	UTILITY PROGRAMS
CHAPTER 7	DIBOL DEBUGGING UTILITY (DDT)
7.1	INTRODUCTION 7-1
7.1.1	Features 7-1
7.1.2	Chapter Organization 7-1
7.2	PREPARING FOR DDT 7-1
7.2.1	CTSGEN 7-1
7.2.2	Compiling 7-1
7.2.3	Linking 7-2
7.2.4	DDT Operation 7-2
7.2.4.1	Running DDT 7-2
7.2.4.2	Failure to Properly Prepare for DDT 7-2
7.2.4.3	Error Messages 7-3
7.3	DDT COMMANDS 7-3
7.3.1	Program Execution Control 7-3
7.3.1.1	Program Execution 7-3
7.3.1.2	Single Step 7-4
7.3.2	Breakpoint Control 7-4
7.3.2.1	Setting Breakpoints 7-5
7.3.2.2	Clearing Breakpoints 7-5
7.3.2.3	Iteration of Breakpoints 7-6

TABLE OF CONTENTS (Cont.)

		Page
7.3.3	Variable Manipulation	7-7
7.3.3.1	Setting Variables	7-7
7.3.3.2	Examining Variables	7-7
7.3.3.3	Extended Variable Manipulation	7-8
7.3.4	Subroutine Traceback	7-8
CHAPTER 8	LINE PRINTER SPOOLERS	
8.1	INTRODUCTION	8-1
8.1.1	Common Characteristics	8-1
8.1.2	Chapter Organization	8-1
8.1.3	Error Messages	8-1
8.2	SINGLE-USER SYSTEMS (LPTSP1.REL)	8-1
8.2.1	LPTSP1.REL Characteristics	8-1
8.2.2	Using LPTSP1.REL	8-2
8.2.2.1	Shared Line Printer Operation	8-2
8.2.2.2	Shared Terminal Operation	8-2
8.2.3	Starting LPTSP1.REL	8-3
8.2.4	Run-Time Dialog	8-3
8.2.5	Response to Error Messages	8-4
8.3	TIME-SHARED SYSTEMS (LPTSPL.TSD and QUE.TSD)	8-4
8.3.1	LPTSPL.TSD Characteristics	8-4
8.3.2	LPTSPL.TSD Requirements	8-6
8.3.3	Running LPTSPL.TSD	8-6
8.3.3.1	Direct Startup (Method 1):	8-7
8.3.3.2	Forced Job Startup (Method 2):	8-8
8.3.3.3	Chain Mode Startup (Method 3):	8-8
8.3.3.4	Implicit Job Startup (Method 4):	8-8
8.3.4	Response to the LPQUE Statement	8-8
8.3.5	QUE.TSD	8-9
8.3.6	Running QUE.TSD	8-9
8.3.7	QUE.TSD Options	8-9
8.3.8	Queing Files for Print	8-10
8.3.9	QUE Commands	8-12
8.3.9.1	ERROR	8-12
8.3.9.2	EXIT	8-13
8.3.9.3	FLUSH	8-13
8.3.9.4	HELP	8-14
8.3.9.5	INTERRUPT	8-16
8.3.9.6	LIST	8-16
8.3.9.7	MODIFY	8-18
8.3.9.8	PURGE	8-19
8.3.9.9	RELEASE	8-19
8.3.9.10	RESERVE	8-20
8.3.9.11	STOP	8-20
8.3.9.12	TALK	8-20
8.3.10	Other Features	8-21

TABLE OF CONTENTS (Cont.)

	Page
CHAPTER 9 PRINTU	
9.1 INTRODUCTION	9-1
9.1.1 Features	9-1
9.1.2 Limitations	9-1
9.1.3 Chapter Organization	9-2
9.1.4 Error Messages	9-2
9.2 THE CONTROL FILE	9-2
9.2.1 COMPUTE (Optional)	9-5
9.2.2 END (Optional)	9-7
9.2.3 EXECUTE (Optional)	9-8
9.2.4 HEAD1 and HEAD2 (Optional)	9-9
9.2.5 IDENT	9-10
9.2.6 INDEX/LIST (Optional)	9-11
9.2.7 INPUT	9-13
9.2.8 OUTPUT	9-17
9.2.9 PRINT	9-18
9.3 PRODUCING THE REPORT PROGRAM	9-20
 CHAPTER 10 ISAM (ISMUTL)	
10.1 INTRODUCTION	10-1
10.2 USING ISMUTL	10-1
10.2.1 ISMUTL Requirements	10-1
10.2.1.1 Single-User Operation	10-1
10.2.1.2 Time-Shared Operation	10-2
10.2.1.3 Error Messages	10-2
10.2.2 Running ISMUTL	10-2
10.2.3 Creating a File (CREATE)	10-3
10.2.3.1 Special CREATE Characteristics	10-3
10.2.3.2 Design/CREATE Process	10-5
10.2.3.3 CREATE Dialog	10-6
10.2.3.4 CREATE Example	10-10
10.2.3.5 Handling CREATE Problems	10-11
10.2.4 Determining the Status of an ISAM File (STATUS)	10-12
10.2.4.1 STATUS Selection and Characteristics	10-12
10.2.4.2 STATUS Example	10-12
10.2.5 Reorganizing a File (REORG)	10-13
10.2.5.1 Special REORG Characteristics	10-13
10.2.5.2 REORG Process and Dialog	10-14
10.2.5.3 REORG Example	10-14
10.2.5.4 Handling REORG Problems	10-15
10.2.6 Exiting from ISMUTL (EXIT)	10-16
10.2.7 Miscellaneous ISMUTL Capabilities	10-16
10.2.7.1 STATUS and REORG in Chain Mode	10-16
10.2.7.2 Auto-CREATE	10-19

TABLE OF CONTENTS (Cont.)

	Page
CHAPTER 11 SORTGEN	
11.1 INTRODUCTION	11-1
11.1.1 Characteristics	11-1
11.1.2 Chapter Organization	11-2
11.1.3 Error Messages	11-2
11.2 THE CONTROL FILE	11-2
11.2.1 DETACH (Optional)	11-4
11.2.2 END	11-5
11.2.3 EXECUTE (Optional)	11-6
11.2.4 INPUT	11-7
11.2.5 KEYS	11-9
11.2.6 OUTPUT	11-10
11.2.7 RECORD	11-11
11.2.8 SPACE (Optional)	11-12
11.2.9 SU (Optional)	11-13
11.2.10 TAGS (Optional)	11-14
11.2.11 WORK (Optional)	11-16
11.2.12 Summary of Control File Commands	11-17
11.3 SORTGEN PROGRAM DEVELOPMENT AND USE	11-18
11.3.1 SORTG	11-18
11.3.2 Compiling with SORTM	11-18
11.3.3 Linking	11-18
11.3.4 Running the Sort or Merge Program	11-19
11.3.5 Example	11-19
11.3.6 SORTGEN in Chain Mode	11-20
 CHAPTER 12 SORT	
12.1 INTRODUCTION	12-1
12.1.1 Characteristics	12-1
12.1.2 Chapter Organization	12-2
12.1.3 Error Messages	12-2
12.2 USING SORT	12-2
12.2.1 Keyboard Level Input	12-2
12.2.2 Application Program Level Input	12-3
12.2.3 Parameter Specification	12-4
12.2.4 Stopping SORT	12-6
12.3 SORT COMMANDS	12-6
12.3.1 ASSIGN (Optional)	12-8
12.3.2 CHAIN (Optional)	12-9
12.3.3 COUNT (Optional)	12-10
12.3.4 DETACH (Optional)	12-11
12.3.5 DIBOL (Optional)	12-12
12.3.6 END	12-13
12.3.7 IDENT (Optional)	12-14
12.3.8 INPUT	12-15

TABLE OF CONTENTS (Cont.)

		Page
	12.3.9 ISAM (Optional)	12-16
	12.3.10 KEYS	12-17
	12.3.11 LOCKCC (Optional)	12-18
	12.3.12 MESSAGE (Optional)	12-19
	12.3.13 OUTPUT	12-20
	12.3.14 PAD (Optional)	12-21
	12.3.15 RECLEN	12-22
	12.3.16 SPACE (Optional)	12-23
	12.3.17 TAGS (Optional)	12-24
	12.3.18 TEMP	12-25
	12.3.19 TTNUM (Optional XMTSD Systems)	12-26
	12.3.20 VERSION (Optional)	12-27
	12.3.21 WORK (Optional)	12-28
	12.3.22 Summary	12-29
	12.4 EXAMPLES	12-32
CHAPTER	13 STATUS	
	13.1 INTRODUCTION	13-1
	13.1.1 Features	13-1
	13.1.2 Chapter Organization	13-2
	13.2 GENERAL	13-2
	13.2.1 Starting and Running STATUS	13-2
	13.2.1.1 Time-shared Systems	13-2
	13.2.1.2 Single-user Systems	13-2
	13.2.2 Errors	13-3
	13.2.3 Options	13-3
	13.3 TIME-SHARED SYSTEMS (TSD AND XMTSD)	13-3
	13.3.1 Option D	13-3
	13.3.2 Option F	13-3
	13.3.3 Option H	13-4
	13.3.4 Option J	13-4
	13.3.5 Option Jx	13-5
	13.3.6 Option Kx	13-6
	13.3.7 Option M	13-7
	13.3.8 Option MA	13-7
	13.3.9 Option Mx	13-8
	13.3.10 Option MKx	13-8
	13.3.11 Option T	13-8
	13.3.12 Option X	13-9
	13.4 SINGLE-USER SYSTEMS (SUD)	13-9
	13.4.1 Option H	13-9
	13.4.2 Option M	13-9
	13.4.3 Option MA	13-10
	13.4.4 Option Mx	13-10
	13.4.5 Option MKx	13-11
	13.4.6 Option X	13-11

TABLE OF CONTENTS (Cont.)

	Page
CHAPTER 14 REDUCE	
14.1 INTRODUCTION	14-1
14.1.1 Characteristics	14-1
14.1.2 Chapter Organization	14-1
14.1.3 Error Messages	14-1
14.2 REDUCE OPTIONS	14-1
14.2.1 Query Mode	14-2
14.2.2 /N Option (No Query)	14-2
14.2.3 /V Option (Version Number)	14-2
14.3 USING REDUCE	14-2
14.3.1 Conventions	14-2
14.3.2 Running REDUCE	14-2
14.4 REDUCE EXAMPLES	14-3
14.4.1 Query Mode	14-3
14.4.2 No Query Mode	14-3
14.4.3 Wildcards	14-4
14.4.4 Version Number Mode	14-4
 CHAPTER 15 MESSAGE UPDATE UTILITY	
15.1 INTRODUCTION	15-1
15.1.1 Features	15-1
15.1.2 Limitations	15-1
15.1.3 Chapter Organization	15-2
15.2 MESSAGE FILE STRUCTURE	15-2
15.2.1 Segments	15-2
15.2.2 Messages	15-3
15.3 UPDATING MESSAGE FILE RECORDS	15-4
15.3.1 Running the Utility	15-4
15.3.2 Update Utility Error Messages	15-5
15.3.3 Terminal Input	15-5
15.3.4 Modes of Operation	15-6
15.3.4.1 ADD a message (Mode 1)	15-6
15.3.4.2 MODIFY a message (Mode 2)	15-6
15.3.4.3 DELETE a segment (Mode 3)	15-6
15.3.4.4 LIST a segment on TT: (Mode 4)	15-6
15.3.4.5 LIST a segment on LP: (Mode 5)	15-6
15.3.4.6 EXIT this session with changes made (Mode 6)	15-7
15.3.4.7 ABORT this session without changes made (Mode 7) ...	15-7
15.3.5 Keyboard Commands	15-7
15.3.5.1 [ESC/A] (up arrow)	15-7
15.3.5.2 [ESC/B] (down arrow)	15-7
15.3.5.3 [ESC/D] (left arrow)	15-8
15.3.5.4 [ESC/E]	15-8
15.3.5.5 [ESC/H]	15-8
15.3.5.6 [ESC/L]	15-9
15.3.5.7 [ESC/M]	15-9

TABLE OF CONTENTS (Cont.)

		Page
15.3.5.8	[ESC/R]	15-9
15.3.5.9	[ESC/S]	15-9
15.3.6	Control File Input	15-9
15.4	EXAMPLES	15-10
15.4.1	Terminal Input Operation	15-10
15.4.2	Control File Input	15-16
APPENDIX A	ISAM FILE CHARACTERISTICS	
A.1	INTRODUCTION	A-1
A.1.1	Features	A-1
A.1.2	Appendix Organization	A-1
A.2	ISAM BASICS	A-2
A.2.1	Data Section	A-2
A.2.2	Index Section	A-2
A.2.3	Handling Added Records	A-2
A.2.4	Summary of ISAM Basics	A-2
A.3	ISAM INTERNALS	A-3
A.3.1	Detailed Structure	A-3
A.3.1.1	Data Files	A-3
A.3.1.2	Index File	A-5
A.3.2	Interrelationships and Tradeoffs	A-7
A.3.2.1	Key	A-7
A.3.2.2	Record	A-7
A.3.2.3	Changing Record and Key Sizes	A-8
A.3.2.4	Group	A-9
A.3.2.5	Overflow Area	A-10
A.3.2.6	Append Area	A-10
A.3.3	DIBOL Statements	A-10
A.3.4	Data Storage	A-11
A.3.5	ISAM File Characteristics	A-14
A.3.5.1	Entries in an Empty File	A-15
A.3.5.2	Entries in a Preloaded File	A-15
A.3.5.3	An Overflow Full Error	A-16
A.3.5.4	Determining that an ISAM File Exists and is Not Empty ..	A-16
APPENDIX B	CTS-300 BUILD PROCEDURES	B-1
APPENDIX C	TIME-SHARED RUN-TIME SYSTEM COMMANDS	
C.1	Running a Program	C-1
C.2	The Attach Command	C-3
C.3	The Copy Command	C-4
C.4	The Delete Command	C-5
C.5	The Directory Command	C-6
C.6	The Rename Command	C-7
C.7	The Type Command	C-8

TABLE OF CONTENTS (Cont.)

			Page
APPENDIX	D	CTS-300 OSSL EXTERNAL SUBROUTINES	
	D.1	FMAC	D-2
	D.2	GLINE	D-4
	D.3	R5ASC	D-5
	D.4	SLICE	D-6
APPENDIX	E	GENERAL NOTES ON I/O	E-1
GLOSSARY		Glossary-1
INDEX		Index-1

FIGURES

Figure	3-1	The DKED HELP Frame	3-4
	3-2	DKED HELPC Display	3-5
	3-3	Special Keypad Keys (VT52) Used for Cursor Control	3-12
	3-4	HELPW Display (Search Wildcard Rules)	3-17
	3-5	RULES Display (Search Parameters)	3-18
	3-6	MODEL Display (Model and Brief Search Parameters)	3-18
	3-7	Special Keypad Keys (VT52) Used for Editing	3-20
	5-1	SUD Memory Allocation	5-8
	5-2	TSD Memory Allocation	5-18
	5-3	Memory Allocation for XMTSD as a Background Program	5-19
	5-4	Memory Allocation for XMTSD as a Foreground Program	5-23
	5-5	BGMAN	5-25
	5-6	XMTSD Capabilities	5-28
	6-1	CTS-300 Operating System Generator (CTSGEN)	6-6
	10-1	ISAM CREATE Flowchart	10-7
	A-1	ISAM File Overview	A-3
	A-2	Data Group	A-4
	A-3	Index Area	A-7

TABLES

Table	4-1	Contents of the Symbol Table	4-5
	4-2	Contents of the Label Table	4-5
	10-1	ISMUTL CREATE Modes	10-21
	A-1	File Control Group	A-8

PREFACE

GOALS

This manual supplies information specific to users of a CTS-300 system. It is not a tutorial manual nor is it purely a reference manual. However, one goal is to supply information so a new user can use the system; another is to provide enough detail to make it easy enough to find, so the experienced user can build and use a system for greater functionality and efficiency.

ASSUMPTIONS

The ability to write simple DIBOL programs is assumed as is the ability to perform basic RT-11 operating system procedures. If you are a new user of CTS-300 without this background, you should refer to the *Introduction to CTS-300 and DIBOL*.

STRUCTURE

This manual is divided into three sections:

Section I Overview of CTS-300

This section will be of primary interest to the new user. It discusses the general capabilities of the system and explains the general system components. The kinds of files the system uses are also covered.

Section II Development

Information you need to build both a working system and programs to run under that system is in this section. This section covers the relationship of the CTS-300 system to the RT-11 SYSGEN and details the CTSGEN procedure. The DKED editor is presented and program compilation discussed in detail.

Section III Utilities

All the CTS-300 utilities (except for CTSGEN and DKED which are covered in Section II) are described in this section.

DOCUMENTATION CONVENTIONS

The DIGITAL Command Language (DCL) format is used for the majority of commands used in examples.

User input is shown printed in red ink.

The carriage return terminator is not shown at the end of command lines; its use is assumed. It is shown, however, where confusion might result if it were missing.

<nnn>

is used to indicate an actual key where nnn is the label or function of that key.

<CR>

is the symbol for the line terminator for terminal input. It causes a carriage return and line feed code combination to be sent to the receiving program and echoed at the terminal.

filespec

is a file specification in the general form:

dev:filnam.ext

where:

dev:	is the mnemonic name of an I/O device.
filnam	is the 1 to 6-character file name, whose first character must be alphabetic.
.ext	is the 1 to 3-character file name extension.

[] indicates optional input.

... (ellipsis dots) indicate optional continuation in the form of the last specified element.

RELATED DOCUMENTATION

Document Title	Order Number
Introduction to DIBOL	AA-P042A-TK
CTS-300 Release Notes and Installation Guide	AA-5697G-TC
DIBOL-83 Language Reference Manual	AA-U066A-TK
CTS-300 System Message Manual	AA-M250B-TC
DECFORM User's Guide	AA-5792D-TC
Introduction to RT-11	AA-5281C-TC
PDP-11 MACRO-11 Language Reference Manual	AA-5075C-TC
RT-11 System User's Guide	AA-5279C-TC
RT-11 System Installation Guide	AA-H376B-TC
RT-11 System Generation Guide	AA-M240A-TC
RT-11 Software Support Manual	AA-H379B-TC
RT-11 Master Index	AA-H380B-TC
RT-11 Programmer's Reference Manual	AA-H378B-TC
RT-11 System Message Manual	AA-5284D-TC
RT-11 System Release Notes	AA-5286E-TC
Guide to RT-11 Documentation	AA-5285G-TC
RT-11 System Utilities Manual	AA-M239Z-TC
RT-11 Mini-Reference Manual	AA-M241A-TC
PDP-11 KED User's Guide	AA-H853A-TC
PDP-11 KED Reference Card	AV-H854A-TC
PDP-11 TECO User's Guide	DEC-11-UTECA-B-D
PDP-11 TECO Pocket Guide	AV-D530A-TK

INTRODUCTION TO SECTION I

Section I contains material to acquaint the new user with CTS-300. Chapter 1 explains the relationship of CTS-300 to RT-11 and briefly highlights some of the major components of CTS-300 and how they relate to one another. Chapter 2 lists some of the more often used RT-11 monitor commands that the CTS-300 user will require. Also covered in Chapter 2 are file conventions and CTS-300 data files.

CHAPTER 1

INTRODUCTION TO CTS-300

1.1 INTRODUCTION

The Commercial Transaction System 300 (CTS-300) is an operating system for the DIGITAL Datasystem 300 series of equipment.

CTS-300 consists of a group of programs, including an RT-11 monitor, RT-11 utility programs, and CTS-300 programs. This group of programs organizes the processor and peripheral devices into a working unit for the development and execution of DIBOL application programs. DIBOL is DIGITAL's Business-Oriented Language and is designed for ease of use in the business community. The result is an operating environment expressly suited to business data processing needs.

The most visible parts of a CTS-300 Operating System are the RT-11 monitor, if it is a single-user system, the CTS-300 Run-Time System, if it is a time-shared system, and the CTS-300 utility programs. All these are outlined in this chapter and explained later in this manual.

This manual is directed toward CTS-300 related subjects. RT-11 components are discussed in complete detail in the appropriate RT-11 documentation.

1.2 MONITORS

A monitor is a master control program that observes, supervises, controls, or verifies the operation of a computer system. It coordinates all activities in a system, including I/O supervision, resource allocation, program execution, and operator communication. There are three monitors associated with CTS-300. These three monitors are supplied by RT-11 and they are:

- Single-Job Monitor (SJ)
- Foreground/Background Monitor (FB)
- Extended Memory Monitor (XM)

1.2.1 SJ Monitor

The Single-Job (SJ) monitor runs in systems with up to 56 K bytes of memory. This monitor has limited capability when compared with the other two, but it has the advantage of being small.

1.2.2 FB Monitor

The Foreground/Background (FB) monitor runs in systems with 64 K bytes of memory and allows access to both the foreground and the background divisions of memory. Foreground/background operation provides a way to share the processor between two programs; one in the foreground and one in the background. For example, during program execution, the FB monitor could run the single-user line printer spooler program in the foreground part of memory for output while it accepts keyboard input in the background part of memory.

1.2.3 XM Monitor

The Extended Memory (XM) monitor supports the same capabilities as the FB monitor and, in addition, this monitor will run in systems with up to 256 K bytes of memory. With the XM monitor, it is also possible to run the DIBOL extended memory time-shared run-time system program in the foreground.

1.3 RUN-TIME SYSTEMS

Run-time systems are a function of CTS-300. The purpose of the CTS-300 run-time system is to provide facilities for interpreting DIBOL code. Additionally, in a multiple-user system, a run-time system provides executive control over the various programs that may be running.

There are three run-time systems:

- Single-User
- Time-Shared
- Extended Memory Time-Shared

1.3.1 The Single-User System

As its name implies, a Single-User DIBOL (SUD) run-time system is for single-user, single-program execution. Single-user programs operate under the direct control of an RT-11 monitor.

The maximum memory available is 64 K bytes.

1.3.2 The Time-Shared System

A Time-Shared DIBOL run-time system (called TSD in the non-extended memory version) is for multiple-user, multiple-program execution. Application programs generated for multiple-user operation run under the control of a time-shared run-time program. This time-shared run-time program, developed by the user with supplied CTS-300 resources, contains the interpretive code that operates like the single-user system explained above and, in addition, exercises control over the DIBOL programs being run by the users. The time-shared run-time program, itself, runs under the control of an RT-11 monitor.

Although it is recommended that the TSD program be run under the SJ monitor, it can run under the FB or the XM monitor. In the latter case, it must always run in the background; and, of course, the extended memory capability is not available, even with the XM monitor.

The maximum memory available is 64 K bytes.

1.3.3 The Extended Memory Time-Shared System

The Extended Memory Time-Shared System (XMTSD) is a special version of the time-sharing program to allow programs to load and execute above 64 K bytes. XMTSD must be used with the XM monitor.

XMTSD can be run in the background or the foreground. Foreground operation makes possible additional capabilities such as allowing the user to submit jobs for background operation and makes available several DCL-like commands. These and other features are discussed in Chapter 5. The DCL-like commands are documented in Appendix C.

1.4 DEVELOPMENT

CTS-300 is supplied to users who are interested in developing application programs to solve specific business problems. Development is divided into two broad categories: program development and system development. Both categories are explained in Section II of this manual.

1.4.1 Program Development

The resources provided for DIBOL program development are the DIBOL editor, DKED; the debugging utility, DDT; the DIBOL compiler, DICOMP; and the object module linker, LINK.

DKED, which operates in either an SUD or an XMTSD system, allows you to create and/or modify source files or data files. KED and KEX are RT-11 editors that can also be used in an SUD system.

The DIBOL compiler accepts source files and produces object files ready for linking. In addition, it provides listings and symbol tables.

DDT, the debugging utility, is documented in Section III of this manual.

Linking is the final step in program development, and this is done under the control of the RT-11 LINK utility. Specific examples of linking are provided, where appropriate, throughout the manual.

1.4.2 System Development

Before program development can be accomplished, both the hardware and software must be matched to the specific needs of the user. In this manual, this is called system development. The RT-11 and the CTS-300 systems must each undergo this process. For RT-11, there is RT-11 SYSGEN and for CTS-300, there is CTSGEN. Chapter 6, System Development, describes the process.

1.5 DIBOL UTILITIES AND PROGRAMS

In addition to the resources already mentioned, there are several programs that are useful, either by themselves or in conjunction with the run-time system. Utilities are described in Section III of this manual.

1.5.1 Supplied Utilities

Line Printer Spoolers

LPTSP1 is the line printer spooler program for single-user systems. LPTSPL and QUE (the terminal interface program) are the pair of line printer spooler utility programs for time-shared systems. Line printer spoolers are used to output a data file to one or more line printers, and are documented in Chapter 8.

PRINTU

PRINTU is a utility that provides a means to generate a simple report program to present information in a data file. PRINTU is documented in Chapter 9.

ISAM (ISMUTL)

ISMUTL is a CTS-300 utility program that creates, reports the status of, or reorganizes ISAM files. ISAM files are data files that are accessed through the indexed sequential access method. ISAM files are discussed in Appendix A. ISMUTL is documented in Chapter 10.

SORTGEN

The SORTGEN utility (previously called SORTG/SORTM) is supplied in the form of two files. SORTG, along with a user-generated control file as input, generates the data division of a DIBOL program which is then compiled with SORTM to produce an object file. The object file is linked to produce the program to perform the sort or merge. SORTGEN is documented in Chapter 11.

SORT

SORT is a sort/merge utility that is faster and easier to use than the SORTGEN utility above. It accepts sort or merge operating parameters either interactively from the keyboard, from a file of commands, or from a SORTGEN control file for previous users of SORTGEN. SORT is documented in Chapter 12.

STATUS

STATUS is a utility program that provides real-time information on active single-user and time-shared jobs and on time-shared run-time system parameters. In addition, while running STATUS, there is an option which allows an active job to be terminated or a message to be removed. STATUS is documented in Chapter 13.

REDUCE

REDUCE is a utility that is used to remove the unnecessary blocks from DIBOL run-time programs that result from the way time-shared programs are linked. REDUCE is documented in Chapter 14.

Message Update Utility (MSGUTL)

MSGUTL is a utility used to maintain the CTS-300 run-time system and utility messages that are contained in the message file ERMSG. MSGUTL is documented in Chapter 15.

1.5.2 Other DIBOL Programs

There are other programs related to CTS-300 systems that could be useful in your application. Among these are DECFORM, DECType, QUILL, and RDCP. DECFORM is an application package that is part of the CTS-300 distribution. DECType is a word processing package for CTS-300 owners. QUILL is a query / report writer. RDCP is a communications package.

CHAPTER 2

BASIC COMMANDS AND FILE CONVENTIONS

2.1 INTRODUCTION

There are basic operating commands and file conventions that must be understood and observed if efficient, trouble-free use of the CTS-300 system is to be obtained. It is not the intent of this chapter to explain all the RT-11 commands or to discuss files in detail, but rather to make you aware of everyday requirements.

2.2 COMMANDS

Special function keys and keyboard commands provide communication with the RT-11 keyboard monitor (KMON). These keys and commands allocate system resources, manipulate memory images, start programs and enable file maintenance. After the RT-11 Operating System has been brought up, any RT-11 keyboard command is legal provided the appropriate software module(s) exists on your system. The readiness of KMON to receive a command is indicated by its period (.) prompt character. For the sake of convenience, this chapter contains some of the more commonly used special keys and commands that are available at the RT-11 Operating System level.

All the commands presented in this chapter are explained in detail in the *RT-11 System User's Guide*.

TSD and XMTSD systems have a series of commands that are available without having to return to the RT-11 monitor level. These commands are discussed in Appendix C.

2.2.1 Special Character Functions

The special functions of some characters on the keyboard that are used at the RT-11 monitor level are listed below. These special functions are obtained by simultaneously pressing the key marked CTRL and the character noted (C, O, S, Q, U, or Z).

CTRL/C

A single CTRL/C aborts the program or transaction, clears task-oriented information, and returns control to KMON.

A double CTRL/C operates the same as a single CTRL/C.

If a program is accepting input and CTRL/C trap is set, then:

A single CTRL/C has no immediate effect but is stored in the input buffer. If the program accepts input later, the single CTRL/C aborts the program or transaction, clears task-oriented information, and returns control to KMON.

A double CTRL/C aborts the program and returns control to KMON.

CTRL/O

A single CTRL/O suppresses terminal output while permitting program execution. A second CTRL/O reenables terminal output. Terminal output is lost between the first and the second CTRL/O.

CTRL/S

A single CTRL/S suspends terminal output. No output is lost.

CTRL/Q

A single CTRL/Q continues terminal output after suspension by CTRL/S. Terminal output resumes from the point at which it was suspended. No output is lost.

CTRL/U

A single CTRL/U erases the entire current line of terminal input. The current line is defined as being the characters between the last line feed (or CTRL/C or CTRL/Z) and the present cursor position.

CTRL/Z

Indicates end-of-file (EOF) when the terminal is used as an input file device.

RUBOUT

This key erases the character preceding the cursor.

2.2.2 Basic Command Rules

The following are general rules that apply to all keyboard commands.

- Keyboard commands can be abbreviated as long as the command remains unique. Throughout the manual optional characters in each keyboard command are noted by brackets [].
- Keyboard command syntax requires a minimum of one space between the command and the first argument.
- Each command line must be terminated by pressing the RETURN key. The system accepts only one command per line.

2.2.2.1 Commands to Allocate System Resources — The following are some of the more common RT-11 commands used to allocate system resources. A brief explanation is given for each. See the *RT-11 System User's Guide* for details.

DATE

The DATE command enters the indicated date into the system and allows you to determine what date (if any) is presently in the system. The system date is assigned to newly created files, new device directory entries, and listing output.

TIME

The TIME command allows you to determine the current time of day as kept by the system, or to enter a new time of day. If the time is entered incorrectly, an error message is generated.

ASSIGN

The ASSIGN command assigns a user-defined logical name as an alternate name for a physical device. Only one logical name can be assigned per ASSIGN command, but several ASSIGN commands can be used to assign different names to the same physical device.

DEASSIGN

The DEASSIGN command removes the logical name(s) previously assigned to a device.

LOAD

The LOAD command makes a device handler resident. Program execution is faster when a handler is resident, even though memory area for the handler must be allocated.

UNLOAD

The UNLOAD command makes previously loaded handlers nonresident, thus freeing the memory they occupied.

SET

The SET command changes device handler characteristics and system configuration parameters.

2.2.2.2 Commands to Start a Program — The following two commands are generally used to start programs from the keyboard. See the *RT-11 System User's Guide* for details.

RUN

The RUN command loads the specified memory image file into memory and starts execution.

R

This command is similar to the RUN command except that the file specified must be on the system device (SY:).

NOTE

Effective with Version 5 of RT-11, if an executable file with a .SAV extension exists on the system device, the file can be executed by simply entering the name of the file in response to the KMON prompt. However, throughout this manual the use of the RUN command is shown.

2.2.3 Monitor Error Messages

Monitor error messages are produced by the RT-11 Operating System. See the *RT-11 System Message Manual* for these messages.

2.3 FILE CONVENTIONS

2.3.1 File Naming

CTS-300 system conventions call for a standard three-character device name. Devices may be assigned logical names which take precedence over physical names. File names consist of one to six characters followed by an optional combination of a period with from one to three characters. Those optional characters and period specify the file name extension and usually indicate the type of the file. If an extension is not specified, most system programs assign default extensions to identify that file. See the following section for common file name extensions for the various types of files.

2.3.2 File Types

There are several kinds of files used in an RT-11/CTS-300 system. These fall into two general categories: system files (monitors, device handlers, command files, and other internal files), program files (source, object, and executable files), and data files. Data files are covered in Section 2.3.3.

As explained in the preceding section, files are assigned an extension that usually indicates the file type and function. Below are listed some of the more common RT-11 file extensions and some extensions unique to CTS-300.

Extension	Meaning
.BAD	file with unreadable blocks
.BAK	backup file
.BAT	batch command file
.COM	indirect command file
.DDF	DIBOL data file
.DBL	DIBOL source file
.DIR	directory listing file
.ISM	ISAM file
.LOG	log file
.LST	listing file
.MAC	MACRO source files
.MAP	output (linker) file
.OBJ	object code file
.REL	relocatable image file
.SAV	memory image file
.SYS	monitor or device handler file
.TMP	temporary file
.TSD	program file linked to run in a DIBOL time-shared system
.TXT	text file

2.3.3 CTS-300 Data Files

There are three kinds of data files used in CTS-300: sequential files, random access sequential files, and ISAM files.

2.3.3.1 Sequential and Random Access Sequential Files — DIBOL sequential files are composed of records containing ASCII characters. As each record is stored, a carriage return / line feed character is automatically appended. When the file is closed (after being opened in output (O) mode) a CTRL/Z character (decimal 026) is written by CTS-300 following the last record. With the exception of multivolume files, discussed below, the CTRL/Z character is required in any sequential file accessed by CTS-300.

Random access requires the records to be of a fixed length.

2.3.3.2 Multivolume Sequential Files — The CTS-300 system can create a logical file that consists of one or more volumes. Each volume is a single, separately-named RT-11 file that can reside on either the same device or a different device. For example, a CTS-300 data file might consist of six volumes: the first three residing on DK0: as DATA1.DDF, DATA2.DDF, and DATA3.DDF; the fourth residing on DK1: as DATA4.DDF; the fifth, DATA5.DDF, residing on a disk pack that is not presently mounted; and the last on DL1: as DATA6.DDF. This arrangement allows files to be virtually unlimited in length.

As with single-volume sequential files, the last record in the last volume of a file is always followed by a CTRL/Z character. The file must have been opened in output mode for the CTRL/Z character to have been inserted upon execution of the CLOSE statement. This CTRL/Z character is interpreted by the CTS-300 run-time system as a logical end-of-file (EOF).

However, by using one of the FLAGS options (see the *DIBOL-83 Language Reference Manual*), multivolume files can be disabled. In this case, for an input file, an EOF will be returned when the last block is read or a CTRL/Z is detected; whichever is first. For an output file, an error message indicating that the output file is full will appear if the space allocated for the file is exceeded. When the output file is closed, the unused portion of the final block is cleared to nulls, but no CTRL/Z is appended.

It is the responsibility of any DIBOL program and the user to distinguish between the volumes of a multivolume sequential file, since neither the CTS-300 run-time system nor the RT-11 Operating System can do so. Each volume is simply another RT-11 file. Further, as far as the run-time system is concerned, each volume is identical except for the one that contains the CTRL/Z character which is detected as an EOF condition. The RT-11 files which comprise the volumes of the logical file may be located anywhere there is room on a disk.

During sequential output (FORMS, WRITES, or DISPLAY statements), whenever RT-11 detects the physical EOF (that is, the last available block is reached) before a CLOSE statement is issued, the run-time system assumes that another volume of the logical file is to be created. The run-time system displays the following message at the console terminal and waits for a response:

```
Mount next output file for dev:filnam.ext
```

where dev:filnam.ext is the file whose physical EOF was just found. If another volume is being created, type in the new file specification. For example, if DATA1.DDF on device DK1: were the first volume and DATA2.DDF on device DK2: were the desired second volume, you would type the following:

Mount next output file for DK1:DATA1.DDF DK2:DATA2.DDF

This new file becomes the next volume in the logical file; the old output file is closed; and the processing continues until either the last available block of the newly created file is used or a CLOSE is issued.

If there is to be no successor to this volume, or if a premature termination is desired, you may respond to the message by typing a CTRL/Z, which causes a full output file error condition. The data that caused the overflow will not be written.

During input (ACCEPT statement with alpha argument or READS statement), if a physical EOF is detected before a logical EOF (CTRL/Z), it is assumed that the logical file continues elsewhere in another volume (RT-11 file). The run-time system displays the following message at the console terminal and waits for your response:

Mount next input file for dev:filnam.ext

where the meaning and response are similar to the message for the output file above. That is, if another volume is available for input, type in the new file specification, and program processing will continue until a physical EOF is again detected or until programming ends. If there is no successor to this volume, or if premature termination is desired, you can respond to the message by typing CTRL/Z, which causes an EOF error condition, or a branch to the EOF label specified in the ACCEPT or READS statement.

When the next volume of the logical input or output file is to be continued on the same device drive with a different disk or tape, the CTS-300 system must wait while the old disk or tape is exchanged for the new one. This is done by including a /W at the end of the file specification which was given in response to the mount successor message. For example, using the above example for an output successor (only this time with the same device) type:

Mount next input file for DK1:DATA4.DDF DK1:DATA5.DDF/W

CTS-300 closes the file on the old unit and displays the message:

Waiting for DK1:DATA5.DDF...

You may now exchange the new disk for the old one. When the new disk or tape is ready, type

<CR> or <LF>

to open the next volume of the logical file on the new disk or tape and continue processing.

When performing input using the ACCEPT statement with a decimal argument, the program operates as described above except that CTRL/Z is treated as any other character, and termination of the file can be accomplished only by a CTRL/Z response to the mount successor message.

When performing random access operations using READ and WRITE statements, the program must choose the correct volume to access. This is because a record in a random access file is located in relation to the beginning of the file on the current volume being accessed, not to the entire logical file that comprises all volumes. In this mode of operation, it is often useful for a number of volumes to be kept open simultaneously, each on a separate channel, so the program can more easily find the volume containing the desired record.

2.3.3.3 ISAM Files — An ISAM (Indexed Sequential Access Method) file is a multivolume file in which the records are stored and accessed according to an index file and a linking scheme that are a part of the ISAM file structure. All volumes of an ISAM file must be loaded and all are open simultaneously. Access to the proper device is automatically maintained.

Aside from storage and access, there are some other differences between an ISAM file and a sequential file. One difference is that in an ISAM file the EOF marker is not a CTRL/Z but rather an entire record in which each bit is set to a one. Another difference is that in a sequential file a carriage return/line feed is appended to each record.

CTS-300 ISAM files are discussed in detail in Appendix A. The ISAM utility ISMUTL is discussed in Chapter 10.

INTRODUCTION TO SECTION II

Section II covers the CTS-300 facilities that are provided for system and program development.

Program development begins with a source file consisting of DIBOL program statements. This file can be created with the DIBOL keyboard editor, DKED, described in Chapter 3. The source file is compiled with the DIBOL compiler, described in Chapter 4. The final step in producing an executable program is linking. Linking is illustrated throughout the manual to show specific requirements and is documented in the *RT-11 System User's Guide*.

Before any program (RT-11 or CTS-300) can be run, however, the total hardware/software system must be configured for its intended use. This is accomplished via an RT-11 SYSGEN and a CTS-300 CTSGEN, both described in Chapter 6. In order to know what the CTS-300 system capabilities are, both for the CTSGEN and for later day-to-day use, the RT-11 Operating System and the CTS-300 Run-Time Systems are discussed in Chapter 5.

CHAPTER 3

DIBOL KEYBOARD EDITOR

3.1 INTRODUCTION

DKED is the text editor supplied with CTS-300. DKED makes it possible to create and edit source files under both Single-User Systems with the Extended Instruction Set (EIS) and Extended Memory Time-Shared Systems. These are files DKED.SAV and DKED.TSD, respectively.

DKED is a program for the CTS-300 system user who has an occasional need for a text editor. It is not meant to be an exhaustive text editor nor will it perform at stand alone speed while there are concurrent programs running.

3.1.1 Requirements and Limitations

DKED CTS-300 requirements are:

- SUD systems require EIS hardware to run DKED.
- XMTSD is the only time-shared system that supports DKED.

3.1.2 Characteristics

Some of the pertinent characteristics of DKED are listed below. They are covered in detail later in this chapter:

- DKED is designed for use with the VT52 terminal. When a VT-100 is used, it is automatically set to the VT52 mode and remains in this mode when you exit from DKED.
- DKED message display is initiated by pressing a key in response to an audible alarm notification.
- A HELP facility is provided.

3.1.3 Chapter Organization

The remainder of this chapter is comprised of four sections. Section 3.2, Operation, discusses preliminary set up requirements for RT-11 and the terminal, DKED terminology, terminal characteristics, running DKED, messages, and the HELP facility. Section 3.3, Creating and Handling Files, presents DKED file characteristics, how to create files and access existing files, and file manipulation commands. Section 3.4, Cursor Control, presents the keyboard functions and commands used to move the cursor. The last section, Section 3.5, Editing, covers text insertion and the keyboard functions and commands used to edit text files.

3.2 OPERATION

3.2.1 Operating System Set Up

If you plan to use a single-user monitor, the RT-11 operating system must be set for your terminal:

```
.SET TT SCOPE
```

```
.SET TT NOCRLF
```

If you are using a VT-100 terminal, select "no line wrap" by setting bit 2 of the third field to 0 using SET-UP B (see the *VT-100 User's Guide*).

3.2.2 Terminology

Some terms used in this chapter have definitions peculiar to DKED. Using DKED is easier if a few of these basic terms are clearly understood at the beginning:

- command A DKED command is a typed entry in combination with the special keypad (see below) command key. Commands are used for file manipulation, to establish search parameters, and to purge the paste buffer.
- function An operation performed by DKED as a result of a specific key (or two-key combination) being pressed. Most functions are selected with the special keypad keys.
- special keypad The small keypad to the right of the main keyboard.
- audible alarm The keyboard tone activated by DKED to inform the user that information is available concerning the present operation (see Section 3.2.6, Messages).
- word A contiguous character string including trailing spaces. A word also includes the operator-inserted carriage return/line feed character pair that terminates a display line.

3.2.3 Terminal Characteristics

DKED operates with either the VT52 or the VT100 terminal. For DKED operation the major difference between the two is in the location of some of the function keys. All the functions described in this chapter are selected by keys on the special keypad for VT52 terminals. For VT100 operation a few of these functions are selected from the main keyboard. These differences are explained in the sections on cursor movement and on editing.

3.2.4 Running DKED

DKED is supplied as an executable program which is run in the same manner as any other program in response to the system prompt.

For Single-User systems:

.R (RU) DKED

For XMTSD systems:

*DKED

DKED indicates its readiness to accept input with an asterisk prompt.

3.2.5 Stopping DKED

To terminate a DKED editing session, the EXIT or QUIT command is used (see Sections 3.3.6.1 and 3.3.6.2). Either of these commands returns you to the DKED asterisk prompt. When the DKED asterisk prompt is displayed, enter a CTRL/C to exit the DKED program and return to the SUD or XMTSD prompt.

3.2.6 Messages

Whenever an error is generated or information is available concerning a particular operation, DKED sounds the audible alarm. When this happens, press the HELP key on the special keypad. This causes a message to be displayed which temporarily replaces the first line of the display. Pressing RETURN, CTRL/W, or any valid function or command will restore the original screen display.

If more information than that contained in the message is needed, refer to the *CTS-300 System Message Manual*.

DKED intermittently displays a "WORKING" message at the top of the screen whenever a command or function is being processed.

3.2.7 The HELP Frame

In addition to its use with messages, the HELP key is used to display general DKED information. When the HELP key is pressed, a HELP frame is displayed. This display (shown in Figure 3-1) shows the keypad layout for the VT52 and summarizes the capabilities and characteristics of DKED. To return to editing, press RETURN, CTRL/W, select a valid function, or issue a valid command.

DKED V06
Keypad Layout for VT52
Lower Function is GOLD

GOLD	HELP	DELIN → UNDELIN	↑ REPLACE
PAGE COMMAND 7	FIND NEXT (Note #1) * FIND 8	BLANK 9	↓ SECTION
ADVANCE BOTTOM 4	BACKUP TOP 5	DELCHR → UNDLCH 6	→ YANK
WORD CHNGCASE 1	EOL DELEOL → 2	CUT * PASTE 3	← APPEND
BEGIN OF LINE		SELECT	ENTER
OPEN LINE		RESET	SUBS

DELETE
CTRL/U
CTRL/W
<GOLD>NNNN

RUB CHAR ←
RUB LINE →
SCREEN
REPEAT

#1) P model = paged search

* = Survives commands
get command list by:
<GOLD><COMMAND> and
respond: "HELPC"

Figure 3-1 The DKED HELP Frame

Summary of HELP frame characteristics

Note 1.

This note in the FIND block reminds you that “\ P \ ” must be specified for a page search (see Section 3.4.10.2).

Asterisk (*)

The asterisk in the FIND and the PASTE blocks indicates two things: the statement “survives command” means the PASTE buffer and search model (see Section 3.4.10) will survive even if you exit from the file and open another file. It also indicates that a display which contains information pertaining to these two commands is available. This information is selected by typing <GOLD> <COMMAND> followed by HELPC. The HELPC response is shown in Figure 3-2.

- | | |
|--|----------|
| 1.) EXIT | |
| 2.) REOPN | |
| 3.) QUIT | |
| 4.) SET SEARCH BEGIN (default) | or:SSB |
| 5.) SET SEARCH END | or:SSE |
| 6.) CLEAR PASTE | or:CP |
| 7.) SET EXACT CASEFLAG | or:SEC |
| 8.) SET EXACT NONE (default) | or:SEN |
| 9.) SETWILDCARDS OFF (default) | or:SWOFF |
| 10.) SET WILDCARDS ON | or:SWON |
| 11.) HELP WILDCARDS | or:HELPW |
| 12.) HELP COMMANDS | or:HELPC |
| 13.) SEARCH RULES | or:RULES |
| 14.) MODEL (displays the current search model) | |

Type <RETURN> to continue:

Figure 3-2 DKED HELPC Display

The HELPC display lists a summary of options and commands:

- Items 1, 2, and 3 show the commands used to manipulate files (see Section 3.3.6).
- Items 4 through 13 show the full and abbreviated commands available for use with search and paste. These are discussed in detail in Sections 3.4.10, for search, and 3.5.14, for paste.
- Item 11, the HELP WILDCARDS (HELPW) command, causes a description of the wildcard rules for search operations to be displayed. See Section 3.4.10.6.
- Item 12, HELP COMMANDS (HELPC), is the command to display the list in Figure 3-2.
- Item 13, the SEARCH RULES (RULES) command, causes a display showing the parameters for the current search. See Section 3.4.10.6.
- Item 14, the MODEL command, causes a display of the current model plus a summary of the current search parameters. See Section 3.4.10.6.

3.3 CREATING AND HANDLING FILES

3.3.1 DKED File Characteristics

DKED is designed to operate on a page basis. A page is a block of text defined by DKED in either of two ways: (1) as the amount of text that can be contained in a fixed percentage (approximately 75%) of the DKED working buffer, or (2) as the area between two form feed characters. The actual size limits (in terms of number of characters) of the buffer and page are a function of several factors, including line length.

As text is inserted and deleted in a situation where the DKED working buffer has reached the page full condition, the position of the page boundaries will shift in relation to the text. Text being inserted after a full page condition causes temporary overflow to the remaining buffer space. After the file is closed the text that overflowed will be placed on the next page. Subsequent editing of this page starts again with a full page condition and with the original unused working buffer space once again available for temporary overflow.

A CTRL/L form feed character, insertable by the operator, can be used to define a page boundary and is displayed as a degree symbol. Page boundaries established by the CTRL/L character remain fixed unless text is added causing a page to exceed DKED buffer limits. Use of the CTRL/L character to mark divisions in your file may make editing easier.

An end-of-file (EOF) marker (not displayed) indicates the end of the last page. When a file is first created it has an end-of-file character in the first character position. This marker is displaced as text is inserted.

The effects of page organization are seen in both cursor movement and in editing and are discussed further in each of these sections. In general, error messages are generated whenever you attempt to exceed a page or buffer limit.

Control Key Functions

Control key functions are selected by pressing the <CTRL> key and simultaneously pressing a letter key.

CTRL/U

Entry of a CTRL/U erases characters to the end of the present line starting at the cursor position. See the DELIN (delete line) function in Section 3.5.3.

CTRL/W

Entry of a CTRL/W restores the screen display following an overwrite by a message or display of the HELP frame.

CTRL/C

DKED ignores a CTRL/C unless at the prompt level; at that level a CTRL/C returns control to either the RT-11 monitor or the run-time system.

CTRL/L

Entry of a CTRL/L is interpreted by DKED as an end-of-page marker once the file has been closed and then opened for further editing. Unlike DKED's end-of-page marker, CTRL/L is visible and appears as the degree symbol.

CTRL/Z

A CTRL/Z is interpreted in either of two ways by DKED, depending on whether it is entered in a command string or entered as a character in a file.

CTRL/Z in a Command String

If you enter a CTRL/Z when DKED is in the command string interpreter mode (a command is being entered in response to DKED's asterisk prompt), CTRL/Z operates like a CTRL/C. That is, DKED operation is terminated and control returns to the RT-11 monitor or the run-time system.

CTRL/Z as a Character in a File

A CTRL/Z can be entered like any other valid character in a file being created with DKED. It is displayed as “ ^ Z”. This allows you to insert this required CTS-300 terminating character in files you create, or it allows you to add it to existing files. DKED interprets a CTRL/Z as an end-of-file (EOF) character in the same way as the nondisplayable EOF character entered by DKED. When a file with a CTRL/Z is closed, the CTRL/Z becomes the EOF and any text following it is lost.

3.3.2 Creating New Files

A new file is created by DKED when you specify the desired file name in the form:

filespec =

where:

filespec is the file specification of the new file. It is in the form dev:filnam.ext.

- dev: is any valid device. The system device is the default.
- filnam is the name of the file.
- .ext is the file extension. The default extension is .DDF.

If the file already exists it will be opened for editing.

NOTE

It is possible to open a file with a null extension by specifying the file name followed by just the dot (.) separator normally between file name and extension (that is, filnam.=).

Error messages are received for the following conditions when creating text files:

- When, under XMTSD, the file exists and is currently in use by another user, you will receive a message that the file is being used and you will not be able to access the file.
- When you attempt to open (create) a file of zero length, you will receive an end message.

3.3.3 Existing Files as Input

Editing Mode

To open an existing file for editing, use the form:

filespec

where:

filespec is the file specification of the new file. It is in the form dev:filnam.ext.

- dev: is any valid device. The system device is the default.
- filnam is the name of the file.
- .ext is the file extension. The default extension is .DDF.

If you want to create another version of an existing file with another name, use the form:

newfilespec = oldfilespec

where:

newfilespec is the file specification of the new file. It becomes the output file upon closing.

oldfilespec is the file specification of the existing file and is the input file.

NOTE

If the input file is a file created with another editor and contains lines whose length are greater than 130 characters, DKED will truncate the lines to 130 characters. No error message is generated. The lines must be shortened before DKED is used or another editor must be used.

Inspection Mode

Occasionally you may want to look at the contents of a text file but do not want to risk changing it in any way. Files opened with the /I switch are available for inspection but are protected from change. Any cursor movement key or command is valid. However, neither the QUIT command nor any of the editing functions are operative in this mode.

To open an existing file for inspection use the form:

filespec/I

where:

filespec is the file specification of the file known to exist. The default extension is .DDF.

/I is the inspection mode switch.

Error Messages

Error messages are received for the following conditions when existing files are used as input:

- When you have specified an input file that does not exist.
- When there is an output file of the same name as the one you specify.
- When, under XMTSD, either the input or output file is currently in use by another user, you will receive a message that the file is being used and you will not be able to access the file.
- When you attempt to open a file that already exists and then respond in a manner that would cause destruction of that file, a message appears informing you that it is an illegal command sequence and requests that you start over.

There are two ways this message could occur:

a) You open (or REOPEN) a file

OUTFIL=INFIL or OUTFIL=

the output file already exists, a QUIT command is issued, and your response is Y (continue even though the file exists).

b) You open (or REOPEN) a file

INFIL or INFIL=INFIL

INFIL.BAK exists and the QUIT command is issued.

3.3.4 Specifying Output File Size

You can allocate a specific number of blocks to a file when it is created or when it is specified as the output file.

Use the form:

filespec[n] =

or

filespecout[n] = filespecin

where:

filespecout (or filespec)
is the file specification of the new file.

filespecin
is the file specification of the input file.

[n] is a decimal number that represents the number of blocks you want to allocate to the output file. The brackets are part of the specification.

Errors are received for the following conditions when output file size is specified:

- when the size specified exceeds the space available on the storage device.
- when you subsequently attempt to input more text into the file than the specified size permits.

3.3.5 File Backup

Whenever a file is closed by DKED using the EXIT command, a backup file is created with the same name as the one just closed but with a .BAK extension. As the file is repeatedly opened, edited, and closed, the backup file is updated. The effect is that upon closing any file there is a copy of the file (the backup file) that remains unchanged until the file is again edited and filed. This prevents loss of data that existed prior to an editing session if a problem should occur during editing.

3.3.6 File Manipulation Commands

There are three file manipulation commands. They are used to close a file, to close and automatically reopen a file, or to abort an edit session.

These commands can be issued at any time during an edit or create session except when a function or command is being processed. Note that the QUIT command, even though it can be issued, can result in unintentional destruction of your file.

File manipulation commands use a combination of the special keypad function keys and typed commands:

Type

<GOLD><COMMAND>

which produces the prompt at the top of the screen:

Command:

to which you respond with the typed command followed by <CR>.

An error message is generated by an improperly structured or invalid command.

3.3.6.1 EXIT Command — The EXIT command is the usual method of terminating an edit or creation session.

The following results from issuance of an EXIT command:

- The file is closed with all edits incorporated.
- If edits were made, a new backup file of the previous version is created.
- If a new output file was specified, it is created with the new name.
- The DKED prompt is displayed.

3.3.6.2 QUIT Command — The QUIT command is used to abort an editing or creating session.

The following results from issuance of a QUIT command:

- All editing done since the file was opened (or since the last REOPN) is canceled.
- The file is closed without change and the output file is purged.
- A message informs you that the purge took place.
- The backup file is not changed or, if there were no backup, none is created.
- The DKED prompt is displayed.

3.3.6.3 REOPN Command — The REOPN command closes a file, reopens it, and places the cursor at the beginning of the file. Since the cursor cannot be moved backward beyond a page boundary, this is a convenient way to position yourself at the beginning of the file from any point in a file.

3.4 CURSOR CONTROL

General

Most cursor control is accomplished using the special keypad function keys. These keys are shown for the VT52 terminal in Figure 3-3. In addition, the RETURN key on the main keyboard also has a cursor movement function.

Some editing functions also result in cursor movement but since they are primarily editing functions they are covered in Section 3.5.

GOLD			↑
PAGE 7	FIND NEXT FIND 8	9	↓ SECTION
ADVANCE BOTTOM 4	BACKUP TOP 5	6	→
WORD CHNGCASE 1	EOL 2	3	←
BEGIN OF LINE			

Figure 3-3 Special Keypad Keys (VT52) Used for Cursor Control

The GOLD key is used to select the lower function printed on a special keypad key.

The VT100 terminal differs from the VT52 in that the arrow keys are located on the main keyboard and the SECTION function (see Section 3.4.9) is selected using the <GOLD><down arrow> sequence.

The cursor can, in general, be moved only to a position already established by some editing function. That is, when a new file is created, the cursor is placed in the first position of the first page followed by the invisible EOF character. No cursor movement is possible under these conditions.

DKED file characteristics have the following effects on cursor movement:

- The cursor cannot be moved backward beyond a page boundary.
- The cursor can be moved to the next page only with the PAGE function or via a search specified to cross page boundaries.

Error Messages

Errors are received for the following conditions related to cursor movement (errors related to the search function are covered in Section 3.4.10.7):

- When you attempt to move the cursor forward beyond a page boundary. The only exceptions are when using the PAGE function, (Section 3.4.8), or a search with the \P\ option, (Section 3.4.10.2).
- When you attempt to move the cursor backward beyond a page boundary.
- When you attempt to move the cursor beyond the boundaries of the file.
- When 0 (zero) is selected for the number of repetitions when using the <GOLD> n <function> sequence (see below).

NOTE

Form-feeds (CTRL/L or degree symbol) or EOF anywhere on the page (and of course there is one on every page) will cause the alarm to sound when the page is initially accessed. The message obtained via the HELP key will state which is the case. This is not an error and normal cursor movement or editing can continue.

Using Keypad Functions

Before any cursor movement is selected, the advance or backup key is used to establish direction.

NOTE

This establishment of direction does not apply to absolute cursor movement functions such as TOP/BOTTOM, PAGE, and the arrow keys - all of which cause immediate movement. In this section the term "absolute" applies to these keys.

A cursor movement function can be repeated by specifying the number of repetitions desired in combination with the GOLD key. The sequence is:

Type

<GOLD> n< function>

where n is the number of times you want the function (cursor movement) to be repeated.

Except for the arrow keys, the <GOLD> n<function> sequence is invalid with absolute functions.

3.4.1 ADVANCE

Causes the next cursor movement to be in the forward (end-of-file) direction. It does not produce movement by itself and does not affect a following request that is absolute in nature.

3.4.2 BACKUP

Causes the next cursor movement to be in the backward (beginning of file) direction. It does not produce movement by itself and does not affect a following request that is absolute in nature.

3.4.3 TOP

An absolute cursor movement function that positions the cursor at the beginning of the current page.

3.4.4 BOTTOM

An absolute cursor movement function that positions the cursor at the end of the current page.

3.4.5 BEGIN OF LINE (Beginning of Line)

Positions the cursor at the beginning of the next line if in advance mode; beginning of previous line if backup mode.

3.4.6 EOL (End of Line)

Positions the cursor at the end of the current line if in advance mode; end of previous line if backup mode.

3.4.7 WORD

Positions the cursor at the first character of the next word if in advance mode; first character of previous word if backup mode.

The beginning of a blank line is interpreted as a word for the purpose of cursor positioning with this function.

3.4.8 PAGE

Positions the cursor at the beginning of the next page.

3.4.9 SECTION

Positions the cursor 16 lines forward if in advance mode; 16 lines backward if in backup mode.

With the VT100 terminal this function is selected with the <GOLD><down arrow> sequence using the main keyboard arrow key.

3.4.10 FIND (Search)

The FIND function searches the text for a match with a character string you specify. Once the FIND function is used, its parameters become the basis of other related functions — FIND NEXT, REPLACE, and SUBS (substitute). Find NEXT is another cursor movement function and is explained in Section 3.4.11. REPLACE and SUBS (substitute) are editing functions and are discussed in Sections 3.5.8 and 3.5.9.

The characteristics of the FIND function are:

- The character string you specify is called a model.
- A search within a page can be in either the forward or the backward direction as established by the ADVANCE or BACKUP key.

- A backward search will not proceed beyond the beginning of the present page.
- A forward search can continue beyond a page boundary only if so specified.
- You can specify whether the cursor is to be placed at the beginning or the end of the character string once the string is found.
- A search can be made for an exact case match.
- Wildcards can be used to allow specified characters to be excluded from the comparison.
- Three HELP frames are available to inform you of search rules and to remind you of the parameters you have chosen for a particular search.

The following subsections describe the features and options of the DKED FIND function.

3.4.10.1 Mode — If you are in advance mode, the search starts from the present position and is done by line, from left to right, in a forward direction. Whether you search beyond the present page depends on how you specify the model. If you are in backup mode, the search is done by line, from right to left, from the present position, in a backward direction. The backward search remains in the present page; it is not possible to proceed past the beginning of the present page in any mode.

3.4.10.2 Model — The model is the search string specifier. It is supplied for the FIND function and serves for the subsequent FIND NEXT, REPLACE, and SUBSTITUTE commands. The model must be changed by another selection of the FIND function.

If the first three characters of the model specification are `\P\` (backslashes), the search will continue to succeeding pages until the target is found, or an end-of-file is encountered. This applies in the forward mode only. As an example of a search, consider the following models:

Model	Meaning
<code>\P\ ABC</code>	Searches for “ABC” in the present page and continues to successive pages until the search is successful or the end-of-file is encountered.
<code>ABC</code>	Searches for “ABC” in the present page and stops when the search is successful or the end of the page is reached.

3.4.10.3 Set Search BEGIN/END — You can select, prior to the search, whether the cursor is to be placed at the beginning or the end of the text string after it is located.

The command sequence `<GOLD> <COMMAND> SSE<CR>` selects positioning at the end of the string.

The command sequence `<GOLD> <COMMAND> SSB<CR>` selects positioning at the beginning of the string. This is the default choice when DKED is first run.

3.4.10.4 Set Exact NONE/CASEFLAG — You can choose to require that the text string specified in the model be an exact match with the target on a case basis as well as a character basis.

The command sequence <GOLD> <COMMAND> SEC <CR> places an exact case restriction on the match for it to be valid.

The command sequence <GOLD> <COMMAND> SEN <CR> allows the match to be made without regard for case. This is the default choice when DKED is first run.

3.4.10.5 Set Wildcards ON/OFF — You can elect to ignore certain characters or character strings in the search match by replacing them with wildcard characters in the search model.

The command sequence <GOLD> <COMMAND> SWON <CR> enables the use of wildcards.

The command sequence <GOLD> <COMMAND> SWOFF <CR> causes wildcard characters to be treated like any other character. This is the default choice when DKED is first run.

The wildcard characters that can be used with DKED are the dot (.) and the asterisk (*), sometimes called a star.

The dot wildcard

The dot (.) indicates that any character in the dot's position will be accepted for a match. That is:

A.B, matches	AXB
	AYB
	AZB

The asterisk wildcard

The asterisk (*) is equivalent to an indeterminate number of dot wildcards and indicates that a match will be made with any character string (including a string of zero length) in the asterisk's position. That is:

A*B, matches	AB
	AXB
	AXXXXXXXB

An asterisk wildcard is useful for finding a target when only the first and last characters are known.

Additional Wildcard rules

- A model cannot begin or end with a wildcard.
- Asterisks and dots can be combined to select a target in which the number of characters in some positions is important and in which the number of characters in other positions is not important.

That is:

A.B*C, matches	AXBC
	AYBC
	AXBXC
	AYBXXXXC

- Adjacent asterisks are allowed; however, the meaning is equivalent to one asterisk.
- Adjacent asterisks and dots force the indeterminate asterisk string to have a minimum length determined by the number of dots.

That is:

A*..B, matches	AXYB
	AWXB
	AWXYB
	AWXYZB

- If wildcards are in use, a search cannot be made for either the asterisk (*) or dot (.) character.

3.4.10.6 Search HELP Frames — The HELP frames available with FIND are selected using the DKED command format: <GOLD> <COMMAND> command <CR>. The following three figures show the screen displays related to search functions:

HELPW

A summary of search wildcard rules are selected with the HELPW command:

Wildcards used in search models:

Dot (.) Don't-care character which matches any single character in the target string. A.B matches AXB, AYB, AZB, etc.

Star (*) Equivalent to an indeterminate number of dots. The star matches any character string less than a line long (including one of zero length). A*B matches AB, AXB, AXXB, AXXXXXXXXXXXXYZB

Notes:

- 1.) Stars and dots may both be used in a model. E.g.: A*B..C*D
- 2.) <Replace> and <Substitute> commands can use models which contain wildcards. Thus the model is an "expression". <Replace> and <Substitute> may have multi-line replacement fields. <Substitute> is repeatable.
- 3.) A model can neither begin or end with a star or a dot.
- 4.) If wildcards are in use, it is not possible to search for an asterisk or a period.

Type <return> to continue:

Figure 3-4 HELPW Display (Search Wildcard Rules)

RULES

This display summarizes the current search options: search begin/end, case none/exact, wildcards off/on, and advance/backward. Also shown are the model, whether the \P\ option is selected, and the length of the model in characters.

Search rules are selected with the RULES command:

```
SET SEARCH BEGIN (or END)
SET EXACT NONE (or CASEFLAG)
SET WILDCARDS OFF (or ON)
ADVANCE (or BACKWARD)
Model: xxx
Paging: No(Yes)
Length: ( n)
```

Type <return> to continue:

Figure 3-5 RULES Display (Search Parameters)

MODEL

This display is a summary of the current search model and associated parameters. It shows the model (xxx), whether the \P\ option is selected, and the model length in characters.

The search model and associated parameters are displayed with the MODEL command.

```
xxx
```

```
Paging: No(Yes) Length: ( n) Type return to continue:
```

Figure 3-6 MODEL Display (Model and Brief Search Parameters)

3.4.10.7 Using FIND — Before requesting a search for a particular text string you usually set up such options as search begin/end, exact case/none, and use of wildcards using the <GOLD> <COMMAND> command <CR> sequence. The search is requested with the following sequence:

Type

```
<GOLD> <FIND>
```

which produces the prompt

```
Model:
```

to which you respond with the desired text string (preceded by the paging option if desired). After you enter the model, pressing the advance or backup key establishes the search direction and initiates the search.

The search begin/end, exact case/none, and use of wildcards can be changed at any time using the <GOLD> <COMMAND> command <CR> sequence.

3.4.10.8 Error Conditions — In addition to the page/file boundary limit errors, messages are received for the following conditions related to the search function:

- No search model
- Invalid use of either the dot or asterisk wildcard (cannot be at the end or the beginning)
- Target not found within the limits of the search

3.4.11 FIND NEXT (Continue Search)

Continues the search using the parameters established in the preceding search. If there were no preceding search, no action is taken. Direction of search is controlled by the direction established in the preceding search.

The search parameters (other than model) can be changed as with the FIND function.

3.4.12 Arrow Keys

The arrow keys are located on the main keyboard for the VT100 terminal and on the special keypad for the VT52 terminal. They cause immediate movement of the cursor in the direction of the arrow.

The arrow keys function with the <GOLD> n<function> sequence to specify repetition.

Up Arrow

Places the cursor at a position in the preceding line directly above the position in the present line. If the length of the preceding line is less than the present line, the cursor is placed following the last character of that line.

Down Arrow

Places the cursor at a position in the following line directly below the position in the present line. If the length of the following line is less than the present line, the cursor is placed following the last character of that line.

Left Arrow

Places the cursor at the preceding character or space. If the present position is at the beginning of the line, the cursor is moved to the end of the preceding line.

Right Arrow

Places the cursor at the following character or space. If the present position is at the end of the line, the cursor is moved to the beginning of the following line.

3.4.13 RETURN Key

The RETURN key on the main keyboard can be classified as a cursor movement function in that it moves the cursor to the beginning of the next line. However, by so doing it is also performing an editing function by inserting characters, creating a new line, and defining the line length. See the next section for related information.

3.5 EDITING

General

Most editing is accomplished using the special keypad function keys. These keys are shown in Figure 3-7 for the VT52 terminal.

GOLD		DELIN →	
		UNDELIN	REPLACE
7	8	9	
		DELCHR →	
4	5	UNDLCH 6	YANK
CHNGCASE 1	DELEOL → 2	CUT	
		PASTE 3	APPEND
		SELECT	
OPEN LINE		RESET	SUBS

Figure 3-7 Special Keypad Keys (VT52) Used for Editing

The GOLD key is used to select the lower function printed on a special keypad key.

The VT100 terminal uses the following keys in place of the keys on the VT52 terminal:

On the VT100 main keyboard

- <GOLD> <up arrow> for REPLACE
- <GOLD> <right arrow> for YANK
- <GOLD> <left arrow> for APPEND

On the VT100 special keypad

- <GOLD> <ENTER> for SUBS (substitute)

There is only one command used in editing. It is used with the PASTE function. See Section 3.5.14.

DKED file characteristics have the following effects on editing:

- In addition to the limitations imposed by the page organization discussed in Section 3.3.1 there is a line length and wrap limitation. A line is defined as a string of characters terminated by a carriage return / line feed pair. This line terminator is entered with the <CR> key. Maximum line length is 130 characters.

If a line of text is accidentally entered which exceeds the screen width, continuation to the next line occurs (preceded by a displayed new-line symbol) and characters can continue to be entered. However, response is very slow. The alarm will sound when the number of characters exceeds 130.

The best way to recover from this situation is to back up using the left arrow key and insert a CR . Do not try to use a deletion function.

NOTE

If the file being edited is a file created with another editor and contains lines whose length are greater than 130 characters, DKED will truncate the lines to 130 characters. No error message is generated. The lines must be shortened before DKED is used or another editor must be used.

- If the DKED text buffer should become filled before a CTRL/L is encountered, the editor sounds the audible alarm and informs you with a message.

DKED allows you to finish the last line you are working on, even if the buffer is full. You must use the PAGE or REOPN command to complete the editing.

- If the end-of-file is encountered, the editor sounds the audible alarm and informs you with a message.
- You cannot return to the previous page (or pages) within the file. You must use the REOPN command to return to the beginning of the file.
- You cannot CUT an area that spans a page break.

Error Messages

Errors are received for the following conditions related to editing functions:

- A full text buffer.
- Cursor not on target for a function requiring a preceding successful search
- Excessive page length. This message indicates that the page is full and that additional text will be placed on another page
- More than 130 characters per record (line)

- No search model for a function (other than FIND) that requires a search model.
- Not enough room in the output file.

Errors related to the search function are discussed in Section 3.4.10.8.

Inserting Text

Text is inserted by typing at the main keyboard. Input starts at the cursor position and consists of characters, spaces, and blank lines. Function keys are used to manipulate text once it has been entered.

Some editing functions can be repeated by specifying the number of repetitions desired in combination with the GOLD key. The sequence is:

Type

<GOLD> n <function>

where n is the number of times you want the function to be repeated and the function is one of those in this section for which repetition is valid.

3.5.1 DELCHR (Delete Character)

Deletes the character at the present cursor position.

Function repetition using the <GOLD> n <function> sequence is valid for this editing function.

3.5.2 UNDLCH (Undelete Character)

Inserts the last character deleted by the DELCHR function.

3.5.3 DELIN (Delete Line)

Operates the same as CTRL/U. That is, it deletes (erases) all the characters from the cursor position to the end of the line including the carriage return / line feed character pair.

Function repetition using the <GOLD> n <function> sequence is valid for this editing function.

3.5.4 UNDELIN (Undelete Line)

Inserts the last character string deleted by either the DELIN or the DELEOL function.

3.5.5 DELEOL (Delete End of Line)

Deletes (erases) all characters from the cursor position to the end of the line excluding the carriage return / line feed.

3.5.6 CHNGCASE (Change Case)

Changes case of the character at the present cursor position and then moves forward or backward for next change (if desired) as previously determined by the ADVANCE or BACKUP function.

Function repetition using the <GOLD> n <function> sequence is valid for this editing function.

3.5.7 APPEND

The APPEND function operates similar to the CUT function except that each selection is added (appended) to the end of text already appended. The first use of APPEND is identical to the CUT function.

With the VT100 terminal this function is selected with the <GOLD> <left arrow> sequence using the main keyboard arrow key.

3.5.8 REPLACE

The REPLACE function is used following a successful search. When the cursor is on the target, the REPLACE function causes the target to be replaced with the contents of the paste buffer.

With the VT100 terminal this function is selected with the <GOLD><up arrow> sequence using the main keyboard arrow key.

3.5.9 SUBS (Substitute)

The substitute function operation is similar to the REPLACE function except the cursor is moved to the next occurrence of the target once the replace is accomplished.

With the VT100 terminal this function is selected with the <GOLD><ENTER> sequence.

Function repetition using the <GOLD> n <function> sequence is valid for this editing function.

3.5.10 OPEN LINE

Inserts a blank line in place of the line containing the cursor. The original line and all those following are moved down one line.

Function repetition using the <GOLD> n <function> sequence is valid for this editing function.

3.5.11 SELECT

The SELECT function identifies the starting point of a CUT (or APPEND) function. Any cursor movement function can be used to place the cursor at the first character of the character string that is to be cut. Once SELECT is chosen, you cannot modify the file before the cut.

3.5.12 RESET

The RESET function deletes an existing SELECT function.

3.5.13 CUT

Once the beginning of the character string has been identified with the SELECT function, move the cursor to the last character of the string and press the <CUT> key. This deletes the character string and places it in the paste buffer.

A cut cannot be made across a page boundary because in moving to the next page the select point is lost.

Up to a full page of text can be cut.

3.5.14 PASTE

Once a character string has been placed in the paste buffer with the CUT function, move the cursor to the position where you want the first character of the string to be placed. Press the <PASTE> key to insert the string at the selected point.

Page boundary and line length limitations apply to pasted text.

You can use the CP command to clear the paste buffer. This is entered in DKED command format as <GOLD><COMMAND> CP <CR> .

3.5.15 YANK

The YANK command results in deletion of the present page.

When YANK is selected, the editor queries you for confirmation before doing the deletion with a message asking if you want to kill the present page.

With the VT100 terminal this function is selected with the <GOLD><right arrow> sequence using the main keyboard arrow key.

3.5.16 RUB CHAR OUT (Rub out)

This key on the main keyboard retains the same function in DKED as it has at the monitor level. This key deletes the character immediately to the left of the current cursor position.

CHAPTER 4

DIBOL COMPILER

4.1 INTRODUCTION

With the release of CTS-300 Version 8, the DIBOL compiler is supplied as two files: SJDBL.SAV which operates with the SJ monitor and XMDBL.SAV operates with the XM monitor. For actual use one of these is renamed DICOMP.SAV. For the remainder of this chapter and throughout the manual DICOMP will refer to the appropriate renamed file. DICOMP accepts source files supplied by the user and creates object files (.OBJ files) that are ready to be linked to produce an executable DIBOL program.

4.1.1 Characteristics

The DIBOL compiler can do the following:

- Read DIBOL source files and produce linkable object files.
- Accept up to six DIBOL source code files.
- Accept several options which govern the nature of the files produced, the listings, and the warning messages.
- Respond to certain compiler directives in the source code. See the *DIBOL-83 Language Reference Manual* for details.
- Detect and report syntax errors.

The DIBOL compiler produces up to four types of output:

- a listing of the source program (.OBJ file)
- a table of variable names used in the program (symbol table)
- a table of label names used in the program (label table)
- a report of the number and type of errors that were encountered during compilation

The object file, the listing file, and the contents of the listing file are optional.

4.1.2 Chapter Organization

The remainder of this chapter is comprised of two sections: Section 4.2, Using DICOMP, which explains how to run DICOMP and how to use the various options, and Section 4.3, Error Messages.

4.2 USING DICOMP

4.2.1 Running DICOMP

Once the appropriate file has been renamed to DICOMP.SAV, DICOMP is executed like any other program.

In a single-user system, DICOMP is entered for execution in response to the RT-11 monitor's prompt. DICOMP responds with an asterisk prompt.

The general format for a single-user system is:

```
.R[U] DICOMP  
*filespecs
```

or,

```
.R[U] DICOMP filespecs
```

If DICOMP is to be used in a concurrent development environment, in addition to renaming XMDBL.SAV to DICOMP.SAV, the procedures detailed in Section 5.7.1 of Chapter 5 must be followed.

The form for filespecs, in any of the above, is:

```
[outfil][,listfil]=infil1[,infil2,...infil6][/S1[/S2.../Sn]]
```

outfil is the output of the compiler in the general form:

```
dev:filnam.ext.
```

- If dev: is not specified, the default device is DK:.
- The default extension is .OBJ.
- If outfil is not specified, no object file is generated.

,listfil is the listing file in the general form:

```
dev:filnam.ext
```

- The default device is DK:. Often the listing file is directed to either the terminal (TT:) or the line printer (LP:) by specifying just the device. This could also be used with just a file name to store the listing on a disk.
- The default extension is .LST.

- If no list file is specified, none is produced; if only an object file is specified, only that file is generated.
- If only a listing file is desired with no object file, specify only a listing file device and file specification preceded by the comma. The comma indicates there is no object file.
- If no listing file nor object file is specified, the only action will be a check for any errors in the input. These errors will be listed on the user's terminal.

infil1[,infil2,...infil6]

are the DIBOL source files supplied as input to the compiler. These files are specified in the general form:

dev:filnam.ext

- If dev: is not specified, the default device is DK: .
- The default extension is .DBL.

S1[/S2.../Sn]

are the compiler options. These are discussed in detail in the following section. If more than one is specified, they are separated by slashes. When multiple options are specified, their order is unimportant.

4.2.2 Command Qualifiers

Command qualifiers restrict or moderate the action specified by the command. The qualifier is appended to the command line with a forward slash (/). The qualifiers are listed below.

/C causes a Cross-Reference (CREF) listing to be generated. This listing indicates, by line number, where symbols are used and defined. (See Section 4.2.4.)

NOTE

If SJDBL.SAV is used, the file CREF.SAV must be on the system device in order to obtain a CREF listing. CREF.SAV is a part of XMDBL.DBL.

/D directs the compiler to include debugging information in the object module. This debugging information is required if the DIBOL Debugging Technique (DDT) is to be used to debug this module.

/G creates a log file of the errors generated during compilation. Whether or not a listing file was selected, the log file also contains the statements that are in error. The log file is placed on the DK: and takes the name of the first input file. This file always has an extension of .LOG. The user is responsible for removing this file when it is no longer needed.

The main purpose of the log file is to provide a place for error information when a compilation is requested by an operator using XMTSD in the foreground. (See Chapter 5).

<code>/L</code>	suppresses the program listing, if one has been specified in the command string. It does not suppress the symbol table, the label table, the CREF listing (if requested), nor the summary line showing the number of errors.
<code>/O</code>	suppresses the output of line numbers as part of the object code and causes other optimizations to occur. The resulting program code is smaller and executes more quickly. Line numbers are required by DDT for single-step execution and breakpoints. If <code>/O</code> is not used, line numbers are generated in the object module and optimization does not occur.
<code>/P:value</code>	specifies the number of lines per page in the listing file. The specified decimal value includes three lines for the top margin, three lines for the header, and three lines for the bottom margin (nine in all). If this qualifier is omitted, the system default of 66 lines per page is used.
<code>/S</code>	generates a symbol table and a label table as part of the listing file.
<code>/W</code>	directs the compiler to suppress warnings during compilation. The number of warnings is still reported.

4.2.3 Specifying Output Files

During the early stages of program development, you may find it helpful to specify no object file which will suppress the production of object files until your source program compiles without errors.

4.2.4 Contents of the Listing File

The listing file consists of the following components: the program listing and, if selected, the symbol table and label table. The contents of each component of the listing file are described in the following paragraphs.

4.2.4.1 The Program Listing — The program listing consists of the original source code with line numbers prefixed to each DIBOL statement. Line numbers are not assigned to the following:

- The START statement
- The compiler directives: `.ENDC`, `.IFDEF`, `.IFNDEF`, `.INCLUDE`, `.LIST`, `.NOLIST`, `.PAGE`, `.TITLE`.
- Continuation lines.
- Blank lines.
- Comment lines.

Line numbers are used in the label table and in the cross-reference listing to identify the location of variable names and label names. In addition, line numbers are useful in debugging (DDT) and error trapping at run time.

For each error detected during compilation, an error message appears in the line-numbered listing. Each message appears immediately after the line in which the error is detected.

4.2.4.2 The Symbol Table — The symbol table contains descriptive information about each variable in the source code. The table consists of four columns with the following headings: Name, Type, Dim (dimension), Size. The information under each heading is described in Table 4-1.

TABLE 4-1
CONTENTS OF THE SYMBOL TABLE

COLUMN HEADING	DESCRIPTION
Name	A list of data field names (RECORD, COMMON, field) used by the program.
Dim	The number of data elements in the field. A dimension of zero is assigned to the data type IMDEF.
Type	The data type of the field. The data type may be alpha (Alpha), numeric (Decimal), or improper definition (IMDEF). If the field is defined in a COMMON statement, its data type identification is preceded with the symbol "C-"; C-Decimal, for example.
Size	The number of characters required to store the field. If the field is an array, the size of one array element is shown.

4.2.4.3 The Label Table — The label table contains descriptive information on each label name and external subroutine name in the source code. The table consists of two columns with the following headings: Name and Line. The information under each heading is described in Table 4-2.

TABLE 4-2
CONTENTS OF THE LABEL TABLE

COLUMN HEADING	DESCRIPTION
Name	The name of each label and external subroutine that is used by the program.
Line	The line number at which the label is defined. Zero is assigned to all improperly defined or referenced label names and external subroutine names.

4.2.3 Standard Listings

If a listing file is indicated in the command string, if the program were not suppressed, and if a symbol and label table were requested, the normal output consists of three segments. These are the program listing, the symbol table segment, and the label table segment with an error report appended to the latter.

4.2.4 CREF Listing

When a listing file is specified with the /C switch in the compiler string, a Cross-Reference (CREF) listing is generated and appended to the end of the compiler listing.

DICOMP normally places a CTRL/Z (^Z) at the end of a DIBOL listing; however, it is not done when a CREF listing is appended. When CREF is specified, a temporary file is opened on the system device.

The CREF listing adds as many as four separate sections to the listing file. A discussion of sections and their functions follows. These sections are not labeled on the listing but appear in the order shown here.

4.2.4.1 COMMON Cross-Reference Table — Lists, in alphabetical order, each of the user-defined COMMON symbols, followed by a series of numbers. The first number, followed by the # symbol, indicates the line number in which the symbol is defined. All other numbers indicate line numbers which reference the symbol. Each page of this section is numbered C-n, where n is the page number in the section.

4.2.4.2 Label Cross-Reference Table — Lists, in alphabetical order, each of the program labels, followed by a series of numbers. The first number, followed by the # symbol, indicates the line number in which the label is defined. All other numbers indicate line numbers which reference the label. Each page of this section is numbered L-n, where n is the page number in the section.

4.2.4.3 Symbol Cross-Reference Table — Lists, in alphabetical order, each of the user-defined symbols, followed by a series of numbers. The first number, followed by the # symbol, indicates the line number in which the symbol is defined. All other numbers indicate line numbers which reference the symbol. Each page of this section is numbered S-n, where n is the page number in the section.

4.2.4.4 External Subroutine Cross-Reference Table — Lists, in alphabetical order, each of the external subroutines that is called, followed by a series of numbers. Each line number indicates a line number in which that subroutine is called. Each page of this section is numbered X-n, where n is the page number in the section.

4.3 ERROR MESSAGES

The errors detected by the compiler are errors of syntax or semantics in specific DIBOL statements. These compiler errors should be distinguished from run-time errors, which are detected during program execution.

For each error detected during compilation, an error message appears in the line-numbered listing. Each message appears immediately after the line in which the error is detected.

Example of an error reported in the listing:

```
      8      ...  
      9      WRITES(1,THE etc  
?DICOMP-E62----Undefined symbol - THE  
.  
.  
.
```

The above example shows an error message. A missing quotation mark makes the word THE appear to be an undefined symbol. A warning message would have a “-W-” in place of the “-E-.”

The compiler error messages are described in the *CTS-300 System Message Manual*.

CHAPTER 5

OPERATING AND RUN-TIME SYSTEMS

5.1 INTRODUCTION

This chapter describes general RT-11 and CTS-300 system characteristics, and the single-user and the multi-user (time-shared) environments. It presents some of the system requirements of these two environments, contains information on preparing programs, and explains how to run programs in the two environments.

5.1.1 General Requirements

Whether yours is a single-user or a multi-user (time-shared) system, there are preliminary requirements that must be met before programs can be run. These requirements are satisfied by properly running the RT-11 operating system generation program (SYSGEN) and the CTS-300 system generation program (CTSGEN). SYSGEN allows you to build an RT-11 operating system software environment that is compatible with your hardware configuration. CTSGEN allows you to build a CTS-300 system that is compatible with both the RT-11 system as structured by SYSGEN and the intended DIBOL program requirements. SYSGEN and CTSGEN are discussed briefly in this chapter and in detail in Chapter 6.

5.1.2 Chapter Organization

The remainder of this chapter is comprised of nine sections. Section 5.2, General System Characteristics, discusses aspects of the RT-11 system pertinent to the CTS-300 user. Section 5.3, The Single-User Environment, describes the operation of a single-user system. The remainder of the chapter is devoted to time-shared operation. Section 5.4, The Time-Shared Environment, is a general discussion of time-sharing requirements and characteristics. Section 5.5, TSD Characteristics, covers the operation of a time-shared system without extended memory. Section 5.6, XMTSD Characteristics (Background Operation), discusses the operation of an extended memory system when run as a background program; Section 5.7, XMTSD Characteristics (Foreground Operation), does the same thing for a system running as a foreground program. Section 5.8, Summary of XMTSD Characteristics, is a brief discussion of operating modes and resulting system capabilities offered by XMTSD. Section 5.9, Terminating Time-Shared Operation (RTEXTIT), explains how to run RTEXTIT.

5.2 GENERAL SYSTEM CHARACTERISTICS

This section discusses aspects of RT-11 and CTS-300 of general interest to CTS-300 users. It is intended to provide background for future use of CTS-300; you may have no immediate or direct use for some of this information.

5.2.1 Foreground/Background Operation

Foreground/background operation is a facility provided by the RT-11 operating system and refers to the ability to alternate between two resident programs.

One of the programs is given priority over the other program. This second program (the background program) is allowed to execute only during times when the first (the foreground program) is waiting for I/O or otherwise temporarily does not require processing. The foreground program operates at a higher priority than the background program, and is always given control at the expense of the background program.

You designate the foreground program by the manner in which you link the program (it must be either a .REL file or a .SAV file with virtual overlays). The foreground program is executed with a special run command.

Characteristics:

- The programs can communicate with each other.
- Either of the programs can access the same file. There is no file integrity maintained between files opened in the foreground and those opened in the background.
- The foreground can voluntarily give up time to the background.
- Programs written for single-user operation can be run only in the background.
- A line printer can be loaded for the exclusive use of either the foreground or the background.

Requirements:

- Foreground/background operation requires 56 KB of memory and the real-time clock.
- The FB or XM monitor is required.

The foreground program is usually devoted to real-time applications and the background to the processing of data. Single-User DIBOL systems use the FB monitor if the spooler is used, and XMTSD depends on it heavily for the ability to run the time-sharing program in the foreground.

Foreground/background operation has no effect on the way programs are compiled or on program code. However, foreground/background operation has an effect on such areas as running programs, devices, linking, I/O to user, and the termination of programs. These areas are discussed below.

Devices:

- With the FB monitor devices must be loaded before executing a program in the foreground.
- With the XM monitor devices must be loaded before any program can be executed in the foreground or background.

Loading

The LOAD command can direct the handler for use by foreground (F), background (B), or both (nothing). Subsequent LOAD commands serve to reassign the handler (it does not have to be unloaded).

If a printer handler is specified by LOAD LP (rather than LOAD LP = F or LOAD LP = B) it is assigned to both foreground and background and the output is interleaved.

Unloading

- You cannot use the UNLOAD command for a device used by a foreground job until you terminate the foreground job and then unload that job.
- There is a special function to remove a terminated foreground job:

UNLOAD F

- Actually both the program and associated handlers can be unloaded together. For example, a foreground program and line printer handler (which was loaded for foreground use) could be unloaded:

.UNLOAD F,LP:

I/O to the user:

All communication to or from the program is identified.

Input

To direct input to a program, enter:

CTRL/F for foreground

CTRL/B for background

Output

Terminal output (if both foreground and background programs are running) is preceded by an identifying symbol:

B> for background

F> for foreground

If the output is a continuation (that is, the program already has identified itself) the identifier is not repeated.

- Your input at the terminal is interrupted by either the foreground or the background if it has a message.
- Printing by both foreground and background programs alternates by line unless the line printer is assigned to either the foreground or background job.

Program Execution:

The procedure to execute a program in the foreground is:

1. Boot the system for FB or XM monitor.
2. Make logical assignments.
3. Load devices.
4. Execute the program using:

FRUN filespec

NOTE

FRUN filespec/BUFFER:n is illegal for use with a program to be run in the foreground.

Program Termination:

An example follows to illustrate termination of a foreground job:

Entry/response:

CTRL/F	;identify input to foreground
F	;response
CTRL/C	;stop the foreground
CTRL/C	;program
B	;response
.UNLOAD F,LP:	;RT-11 monitor prompt and command to unload ;the foreground job and handler

5.2.3 Using the Error Logger

The RT-11 error logger and queue program should ideally be running all the time. However, the error logger file could seriously infringe on usable storage space. Since the error logger handler itself requires memory, there might not be enough space left for background operation with XMTSD in the foreground (see Section 5.7).

5.2.3 Using the Error Logger

The RT-11 error logger and queue program should ideally be running all the time. However, the error logger file could seriously infringe on usable storage space. Since the error logger handler, itself, requires memory, there might not be enough space left for background operation with XMTSD in the foreground (see Section 5.7).

Realistically, the use of the error logger should be confined to situations where you have known problems. It is suggested that you run it in a system that has been configured (via SYSGEN) for tasking or that you run it as a foreground program.

5.2.4 Utilizing Resources on a Small System

The relatively limited capacity and access speed of a diskette, especially when used as the system device, make careful planning necessary if you want to make the best use of your system. The following guidelines are presented to make you aware of what may be done:

- Include only the essential programs, data files, utilities, and options.
- Use the SEGMENT:1 option when initializing diskettes prior to building your system. The result is a two-block directory on each diskette. This gives you a directory with a 72-file capacity and saves six blocks (two blocks per segment) of diskette space.
- If you have an application program that uses overlays, consider relinking your overlaid routines. Where possible, make them a part of the root section of your program. This reduces disk access time. Programs that use overlays heavily are slower on a diskette system than when run on a system with faster disks. The elimination of overlays will, therefore, increase performance.
- If you set USR to NOSWAP, you reduce the time to open and close files by 90 percent. This is recommended as a standard procedure if you plan to be manipulating files extensively and if you have the space in memory (the USR requires 4 KB).
- Depending on the size of your application and the functions you want to support, you might segregate programs to be used together and place them on a single diskette. That is, not all the capabilities need be placed on the system disk but could be grouped by logical use onto individual diskettes. If you swap disks in this way to perform various functions, you also have to set USR to SWAP so the directory lookups for files will work correctly.

5.2.5 Stopping Programs

A program normally continues until it runs to completion and a STOP or END statement is executed. You can prematurely terminate program operation by typing a CTRL/C at the keyboard of the terminal to which the program is currently attached. The CTRL/C function can be disabled by the CTRL/C trap which is requested via CTSGEN or an XCALL to the FLAGS subroutine at run time. If you are planning to use CTRL/C to terminate program execution, it is wise to be certain the trap is not set. If the program is running in a detached state, the ATTACH command must first be used to connect the terminal to the program before CTRL/C can be used to stop it. When the program stops, control returns to the run-time system.

CTRL/C normally causes immediate program termination. If the program is not waiting for keyboard input (that is, executing a READS statement to the terminal), two CTRL/Cs typed in succession are needed to terminate program operation. Files opened for output (O) mode are lost since they cannot be closed.

Following are some of the implications of the CTRL/C command:

- If keyboard input is pending, or if no I/O operation is in process, CTRL/C stops the program immediately.
- If an I/O operation is in process, CTRL/C stops the program only after the I/O operation is completed.
- Two CTRL/Cs in succession will cause an immediate, unconditional stop, regardless of any I/O operation that may be pending.
- CTRL/C has the following effect on files: Files opened in the output (O) mode are lost, since they cannot be closed. Files opened in the update (U) mode may not have all changes incorporated in them.
- The CTRL/C command can be disabled (or enabled) via CTSGEN or, from within a DIBOL program, by means of the external subroutine FLAGS. See the *DIBOL-83 Language Reference Manual*.

For information on unloading programs and handlers in the foreground once a program has been stopped, see Section 5.2.1.

5.2.6 CTS-300 System Function

The function of any CTS-300 system is to allow you to run DIBOL programs. It provides the interface between a DIBOL program and the RT-11 operating system. Although there are some special time-shared related instructions, the particular run-time environment is not a major consideration when writing a DIBOL program. A program becomes a single-user or a multi-user program as a result of how it is linked, not as a result of how it is written.

5.3 THE SINGLE-USER ENVIRONMENT

5.3.1 System Requirements for SUD

For SUD programs you can generate an RT-11 system (SYSGEN) for either the SJ, FB, or the XM monitor. The SJ monitor is recommended unless you intend to use the SUD print spooler (LPTSP1.REL) or intend to use foreground/background operation; then the FB or XM monitor must be used.

The DIBOL compiler produces interpretive code. Therefore, the requirement for DIBOL programs to run under the RT-11 operating system is that there be a run-time code interpreter available. This interpreter, which can be thought of as an executive or a run-time system, is produced as a result of the CTSGEN process. It is assigned the name SUD.RTS by CTSGEN and must always reside on the system disk.

If you plan to use either ISAM or the DIBOL debugging program, DDT, you must make the appropriate selections in CTSGEN.

The single-user run-time code interpreter requires approximately 14.2 KB of memory without ISAM or DDT.

5.3.3 SEND/RECV for SUD

Effective with Version 8, more than one message may be sent by a given program and messages can be sent to any program to be run subsequently.

Messages are stored on a special file on disk. This file (MSGPKT.REL) is created to contain a message at the time the message is sent. Subsequent messages are also placed in this file. MSGPKT.REL exists until all the messages have been "received" out of it. That is, this file exists only when there are messages pending receipt.

The file is automatically deleted when the last message is received or killed (via STATUS). The maximum size of the file is 30 blocks. The first block is an index containing the storage information for each message. The remaining blocks are used to store the messages; a minimum of one block per message is used. If a message needs more than one block (512 bytes) of storage, a search is made of the remaining free blocks until the necessary number of contiguous free blocks is found. If not enough free blocks are available, an error message is generated.

NOTE

If all the messages are not received by a SUD application, the message file will still exist and, if the SEND statement did not specify a name, a RECV by any program could access a message by mistake. Deletion of MSGPKT upon system boot would be one way to eliminate this possibility.

5.3.3 Preparing Programs

Program editing and compilation of DIBOL programs for SUD systems are performed using any valid editor (EDIT, EDT, K52, DKED, etc.) and the DIBOL compiler. DKED and the DIBOL compiler programs are described in Chapters 3 and 4 of this manual.

5.3.4 Linking Programs

Linking is discussed in detail in Chapter 11 of the *RT-11 System User's Guide*. This section contains information specific to a DIBOL single-user run-time system.

The object (OBJ) file resulting from compilation must be linked to the DIBOL libraries (ODIBOL.OBJ and DIBOL.OBJ). The following command string could be used to link the program TEST.OBJ and subroutines SUB.OBJ and SUB1.OBJ for SUD operation:

For programs without DDT:

```
.LINK TEST,SUB,SUB1,ODIBOL,DIBOL
```

results in a SUD program named TEST.SAV.

For programs compiled for DDT:

NOTE

If you plan to use DDT, you must compile with the debugging option.

The following command string illustrates linking for DDT:

```
.LINK TEST,SUB,SUB1,TSDDT,ODIBOL,DIBOL
```

5.3.5 Running Programs

DIBOL programs in a SUD system are run like any other Save file:

```
.R PROG
```

or

```
.RU dev:PROG
```

The run-time interpreter SUD.RTS is automatically brought into memory whenever a single-user DIBOL program is run.

5.3.6 SUD Memory Allocation

The allocation of memory for a SUD system is shown in Figure 5-1.

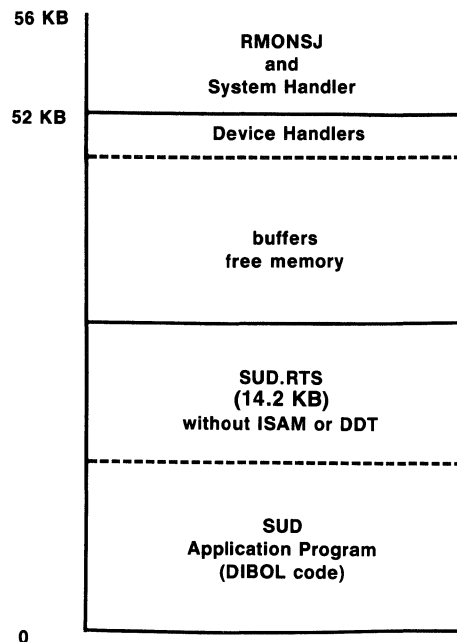


Figure 5-1 SUD Memory Allocation

5.4 THE TIME-SHARED ENVIRONMENT

A DIBOL time-shared run-time system or executive is a program that runs under the control of the RT-11 monitor. Its purpose is to control the scheduling and time-sharing of DIBOL user programs. There are two versions available:

- The normal version, identified as TSD, runs under the RT-11 Single-Job (SJ) monitor, the Foreground/Background (FB) monitor, or the Extended Memory (XM) monitor. Of course, TSD cannot access extended memory even with the Extended Memory monitor. The SJ monitor is recommended because its small size leaves more memory for user programs.
- The extended memory version, identified as XMTSD, runs only under the RT-11 Extended Memory monitor.

A time-shared run-time system permits properly compiled and linked DIBOL programs to be independently loaded and executed in a time-shared environment. The time-shared system communicates with all active terminals to provide interactive program control. Simple keyboard commands, explained below, permit you to initiate and terminate program operation and to perform other routine functions. During execution, each program appears to have at its disposal the full resources of the CTS-300 system. Further, programs can share data files and I/O devices. The time-shared system provides complete facilities for:

- Program loading and execution
- Processing of keyboard commands for routine operations
- Allocation of memory resources
- Program scheduling
- Detached program operation
- Error detection and reporting
- Sending/Receiving messages between programs (see the *DIBOL-83 Language Reference Manual*)
- Line printer spooling (see Chapter 8)

The information in this section (5.4) applies to both TSD and XMTSD systems.

5.4.1 System Requirements for Time-Sharing

As with the single-user system, the DIBOL programs in a time-shared environment require a run-time code interpreter, and CTSGEN produces this code. However, the CTSGEN for a time-shared system also includes the specification (and resulting construction) of the code to process the time-shared related statements. If you plan to use either ISAM or the DIBOL debugging program, DDT, you must make an appropriate selection in CTSGEN. In addition, terminals to be recognized by the run-time system are selected in CTSGEN. Special SYSGEN requirements are presented under the TSD and XMTSD sections (Sections 5.5 and 5.6).

The following table indicates the approximate memory requirements of the two commonly used RT-11 monitors and the two CTS-300 time-shared run-time systems. The time-shared systems both have multiterminal support. The sizes of ISAM and DDT are also shown.

Time-Shared System Component Sizes

Component	Approximate Size
SJ Monitor	7 to 9 KB
XM Monitor	14 to 18 KB
SUD Interpreter (SUD.RTS)	14.2 KB
TSD Run-Time System	22 KB
XMTSD Run-Time System	36 KB
ISAM	4.5 KB
DDT	1.2 KB

5.4.2 Dynamic Memory Allocation

The time-shared run-time system dynamically manages the allocation of the free memory used by DIBOL programs. This means that memory space is allocated automatically on a demand basis. When you request that a program be executed, the run-time system loads the program into the first free area of memory that it finds large enough to contain the program. When a program terminates execution, the memory space that it occupied becomes available for reallocation to another program. When free memory becomes fragmented to the extent that no contiguous area of memory is large enough to contain the program to be loaded, the run-time system attempts to relocate currently existing programs to make the necessary room. The free memory spaces are concatenated and, if there is then enough space, the program is loaded.

Memory is required for buffers whenever an I/O channel is opened by a program. The size of the buffer for each channel opened is determined by the PROC or OPEN statement. Furthermore, memory is allocated to a program by the run-time system in increments of 2 K bytes only. Therefore, a requirement for a block of memory could result in allocation of 2 K bytes if the requesting program does not currently have the memory available. This memory is not released to the system as channels are closed but remains available for future use by the program as other channels are opened.

In a time-shared system, memory is required for tables associated with opening a file. In a TSD system, this memory space is allocated at run time. In an XMTSD system, this space must be preallocated. The answer to the CTSGEN question (see Chapter 6) asking for total number of files to be opened for update is used as the basis for this preallocation. The amount of memory used can be approximated from the information in Chapter 6.

5.4.3 Scheduling

To effect time-sharing operation, the run-time system performs program scheduling (time-sharing) on the basis of real time, I/O request, or a SLEEP statement. This means that a program will run for a predetermined time or until an I/O statement (READ, WRITE, etc.) is executed or until a program issues a SLEEP statement. When either of these conditions occurs, the run-time system suspends execution of the current program and schedules another program to run.

Program execution time scheduling can be controlled within a DIBOL program by means of the SLICE external subroutine. See the *DIBOL-83 Language Reference Manual*.

5.4.4 Detached Program Operation

A program that is running under the control of a time-sharing system program, and that does not require the constant use of a terminal, can disconnect itself from the terminal by using the DETACH statement. Detached operation disconnects a program either from the terminal that initiated the program's operation or from the terminal to which it was last attached. A program such as a printer spooler is a good candidate for detached operation.

After a DETACH statement is executed by a program, the message shown below is displayed:

```
DETACHING  
  
TSD VERSION RT11---TSD-V08.000  
* or  
XM-TSD VERSION RT11---XMT V08.000  
*
```

At this point you may issue either a RUN command or an ATTACH command. See Appendix C.

The detached program runs to completion as long as either no further terminal I/O is required or no error condition occurs that requires a displayed message. If terminal I/O is required, the program's operation is suspended until an ATTACH command is issued from an active terminal.

5.4.5 Data File Management

Files created in the time-shared environment are compatible with those created in the single-user environment and conform to the file conventions discussed in Chapter 2 of this manual. Under the time-shared run-time system these data files can also be shared by two or more programs, thus providing the capability of creating a common data base. In addition, programs can also share certain I/O devices with one another.

5.4.5.1 File Sharing — All programs that are to share a file must open the file with the OPEN statement in either update (U) or in input (I) mode. When a program opens a file for either update or input, the file can subsequently be opened by other programs (in I or U modes). For example, one or more programs can be reading a file sequentially (READS statement) while other programs can be directly accessing the file (READ or WRITE statement).

When a record is read using the direct access READ statement in update mode, the blocks within which the record wholly or partially resides are automatically locked. This means that the record cannot be read by another program in U mode until it is released. Also, other adjacent records may be locked if they happen to reside wholly or partially within the block(s) that contains the record being read. An attempt to access (READ or WRITE statement) a locked record will cause an error message to be displayed. The block(s) that contains the record is unlocked when the program that originally read the record does one of the following:

- Rewrites the record.
- Reads another record from the same file.
- Issues an UNLOCK statement. (When a program that accesses a file using two or more channels issues an UNLOCK, the lock condition is cleared for all channels.)
- Issues a CLOSE to the channel.
- Terminates operation.

Information on use of the OPEN statement:

SU Mode:

Certain applications require that if an ISAM file is to be opened in update (SU) mode it must first exist and, secondly, that it not be empty. The following procedure is used:

```
ONERROR NOFILE
OPEN (ch,SI,'FILE.ISM')      ;does the FILE'ISM exist and
CLOSE ch                     ;is it an ISAM file
OFFERROR
OPEN (ch,I,'FILE.ISM')
```

By opening the indexed file in Input mode, the FCG can be examined to determine the current number of records in the file.

The above method can result in a problem in a time-shared system if two or more programs carry out the above procedure. A file table entry in the run-time system keeps track of the current number of users of the file. The value of this entry will likely be incorrect under these conditions. The problem is caused first by opening the indexed file in Input mode rather than as an ISAM file, and, ultimately, by specifying the same device name in each of the OPENS.

If it is necessary to open an indexed file in Input mode to read the FCG, do the following: open the indexed volume in Input mode, but specify a device name different than the device name specified when opening the file as an ISAM file. The different device names must logically be the same device.

U Mode with multi-channel access:

A DIBOL program should not open a file in update mode on two (or more) separate channels. Simultaneous record locking information cannot be maintained for the two channels when the program reads a record on each of the two channels. This is because there is only one lock table entry per update file per program.

For a sequential file, only the last of the two records read will, therefore, be locked.

For an ISAM file, if the two records are read, updated, and then written, it is possible that the first record that is written will reside in the second record's position in the file. In addition, when the second record is then written, error number 53 (Key not same) will be generated.

5.4.5.2 Device Sharing — Mass storage direct access devices, such as disks, are sharable. Sequential access devices (magtape) are nonsharable devices. Line printers and other non-mass storage devices are also nonsharable.

There is a restriction on the use of the **ASSIGN** command when devices are being shared. If the **ASSIGN** command is used to assign a logical name to a disk, then this assigned name must be unique throughout all programs that refer to that device. That is, the device must be referred to by its assigned name and that name only. To do otherwise creates an ambiguous condition for the run-time system with the following results:

- If two programs attempt to open (**OPEN** statement) the same file, in **U** mode, using different logical device designations, the run-time system will not be able to detect a file contention situation (a record lock error) that should be detected.
- The automatic lock feature will not be invoked to prevent simultaneous reading and writing of the same record.

For information on devices and device handler loading relative to programs running in the foreground, see Section 5.2.1.

5.4.6 Preparing Programs

A program that runs under a time-shared run-time system is identical to a program that runs in the single-user environment except that certain language statements become operable. Specifically these are: **READ**, **WRITE**, **READS**, **WRITES**, **SEND**, **RECEIVE**, **DETACH**, and **SLEEP**. The **SLEEP** command, while it operates in a single-user system, is primarily a time-sharing statement. Details on the use of these statements are provided in the *DIBOL-83 Language Reference Manual*.

All program preparation operations, including compilation and linking, are identical for **TSD** or **XMTSD** run-time systems.

Program editing and compilation are performed using any valid editor (**EDIT**, **EDT**, **K52**, **DKED**, etc.) and the **DIBOL** compiler. **DKED** is the only editor you can use with a time-shared run-time system. **DKED** and **DICOMP** are described in Chapters 3 and 4 of this manual.

If you plan to use **DDT**, you must include the **DDT** option switch in the compiler command string.

5.4.7 Linking

Linking is discussed in detail in Chapter 11 of the *RT-11 System User's Guide*. This section contains information specific to a **DIBOL** time-shared run-time system.

5.4.7.1 Linking to the Time-Shared DIBOL Library — The object (OBJ) file(s) resulting from compilation must be linked with the time-shared run-time system libraries (ODIBOL.OBJ and TDIBOL.OBJ). The following command string could be used for time-shared operation to link the program TEST.OBJ and subroutines SUB.OBJ and SUB1.OBJ:

```
.LINK/EXE:TEST.TSD TEST,SUB,SUB1,ODIBOL,TDIBOL/BOT:100000
```

It is recommended that the file name extension .TSD be used for the program file output by the linker. This results in two important benefits:

- A file linked for time-sharing operation can be easily distinguished from a file linked for single-user operation. This naming convention also prevents a .SAV file from being overwritten if the time-sharing file has the same file name.
- No extension need be specified in the RUN command for a time-shared program with the .TSD extension.

To save disk storage space, the utility program REDUCE should be used to remove the unused blocks resulting from the linking process. See Chapter 14.

5.4.7.2 Linking for DDT Use — If the DDT utility program is to be used with a program running in the time-shared environment, the file TSDDT.OBJ must be included in the link command string. The following command string, assuming the program was compiled for DDT, could be used to link (for DDT operation) the program and subroutines shown in the previous example:

```
.LINK/EXE:TEST.TSD TEST,SUB,SUB1,TSDDT,ODIBOL,TDIBOL/BOT:100000
```

- The file TSDDT must precede the file TDIBOL in the linker's command string.
- DDT operation requires the time-shared system to be a version in which DDT was selected during CTSGEN. See Chapter 6 in this manual.

In order to save disk storage space, the utility program REDUCE should be used to remove the unused blocks resulting from the linking process. See Chapter 14.

5.4.8 Creating Overlays

Programs and their subroutines that are to be run in a time-shared environment can be formed into overlays using the same procedures used with any other program. See the *RT-11 System User's Guide*. The main object module, ODIBOL, TDIBOL, and TSDDT (if used) must all reside in the root segment of the overlay.

The example below illustrates linking for overlays using the same modules as with linking for DDT, above. In this example, subroutines SUB and SUB1 are linked for non-simultaneous operation in overlay region one. The remaining modules are in the root segment.

```
.LINK/PROMPT/EXE: TEST.TSD TEST,TSDDT,ODIBOL,TDIBOL/BOT:100000  
*SUB/0:1  
*SUB1/0:1  
*//
```

5.4.9 Time-Shared Keyboard Commands

A CTS-300 time-shared system accepts commands to perform the following routine functions: execute a program, attach a program operating in the detached state, copy files, delete files, rename files, display files, and obtain a volume directory. These commands are documented in Appendix C of this manual. There are also three commands available only to XMTSD and only when it is operating in the foreground. These additional commands are SUBMIT, SHOW, and CANCEL. They are related to foreground / background communications and are discussed in Section 5.7.6.

5.4.10 Programmed Startup

Time-shared programs can be executed with the RUN command discussed above, or their execution can be initiated by a program which is running under a time-shared system. There are three methods by which this can be accomplished: forced job startup, chain mode startup, and implicit job startup.

5.4.10.1 Forced Job Startup — A forced job startup occurs when the XCALL statement is used with the RUNJB subroutine in the manner presented here. Forced job startup is used to start up a program that is not already running.

The format is:

```
XCALL RUNJB ('filespec',n)
```

where:

filespec is the file specification for the program to be started up.

n is a number with the following meaning:

- If n is positive (including 0), it is the number of the terminal to which the program is to be attached.
- If n is -1, it indicates that the program is to be started up in a detached state.

If the program is started up detached and it requires a response, it is necessary to supply this response with a SEND statement prior to the XCALL for the forced startup. The general form is:

```
SEND ('msg', 'filespec')  
:  
:  
:  
XCALL RUNJB ('filespec',-1)
```

where:

msg is a message sent to the program before it is run. This message is required for some DIBOL programs.

filespec is, in both lines, the file specification for the program to be started up.

-1 indicates the program is started in a detached state.

5.4.10.2 Chain Mode Startup — Chain mode is used when it is desired to execute a program following the normal termination of a first program.

The format is:

```
STOP 'filespec'
```

where:

filespec is the file specification for the desired next program to run.

If the calling program were detached, the program started in chain mode will also be detached. If this second program requires an operator response, it is necessary to supply this response with a SEND statement prior to chaining. The general form is:

```
SEND ('msg', 'filespec')  
.  
.  
.  
STOP 'filespec'
```

where:

msg is a message sent to the program before it is run. This message is required for some DIBOL programs that would normally query the operator.

filespec is, in both cases, the file specification for the program to be started up.

5.4.10.3 Implicit Job Startup — Implicit job startup is used to send a message to a program and to ensure that the program will be automatically started up detached if not already running.

The form is:

```
SEND ('msg', 'filespec', n)
```

where:

msg is the message to be sent.

filespec is the file specification for the program to be started up.

n has a value of -2 or -3:

- A -2 indicates that if the program is not currently running, the time-shared run-time system will start up the job in a detached state.
- A -3 indicates that a copy of the program will be started up in a detached state. The copy started is in addition to any others that may be currently running.

5.5 TSD CHARACTERISTICS

This section applies only to the nonextended memory time-sharing run-time system identified as TSD.

5.5.1 System Requirements for TSD

For TSD you must generate an RT-11 system (SYSGEN) for either the SJ, FB, or the XM monitor. However, even if you use TSD with the XM monitor, you will not be able to access extended memory. The SJ monitor is recommended because of its smaller size.

5.5.2 Running the TSD Run-Time System Program

Once the TSD run-time system has been properly built via CTSGEN, time sharing can begin. Load and execute the time-sharing program (TSD run-time system) by using the monitor's RUN command in the form:

```
.R filnam.ext or .RU dev:filnam.ext
```

where:

dev: is the name of the device where the TSD run-time system program resides. If not specified, the system device is assumed.

filnam.ext is the name and extension of the TSD run-time system. The name is the one specified in answer to the last question in CTSGEN. See Chapter 6. Regardless of the file name you choose, TSD will always identify itself as TSD.

Once started, the TSD run-time system identifies itself by displaying the following message at each of the terminals that it recognizes:

```
TSD VERSION RT11 TSD V08.000  
*
```

where:

n is the version number.

(*) the prompting character, indicates readiness to accept a time-shared keyboard command (see Section 5.4.9).

5.5.3 TSD Memory Allocation

The allocation of memory for a TSD System is shown in Figure 5-2.

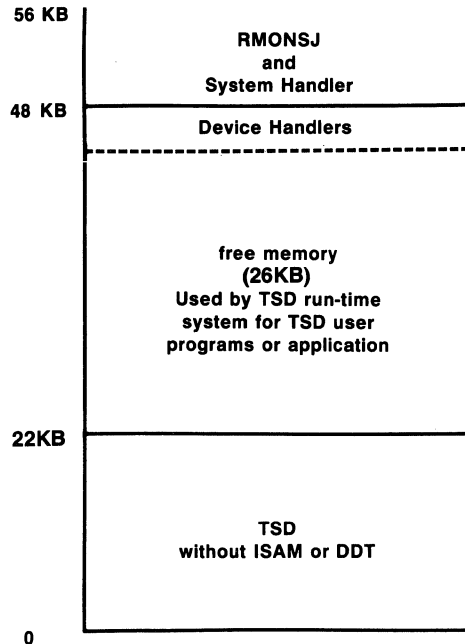


Figure 5-2 TSD Memory Allocation

5.6 XMTSD CHARACTERISTICS (BACKGROUND OPERATION)

XMTSD run in the background has the same capabilities and limitations (except, of course, for the greater memory capacity) as TSD running with the SJ monitor.

5.6.1 System Requirements

XMTSD requires that the XM monitor be selected during RT-11 system generation (SYSGEN). There are no other special requirements, nor are special questions asked for building the XMTSD system.

5.6.2 Running XMTSD in the Background

Once the XMTSD program has been properly built via CTSGEN, time sharing can begin. Load all handlers to be used and then load and execute the time-sharing program (XMTSD run-time system), which must be on the system device, by using the monitor's RUN command in the form:

```
.R filnam.ext
```

where:

```
filnam.ext
```

is the name and extension of the XMTSD run-time system as specified in answer to the last question in CTSGEN. See Chapter 6.

Once started, the XMTSD run-time system identifies itself by displaying the following message at each of the terminals that it recognizes:

```
XM-TSD VERSION RT11 XMT V08.000
*
```

where:

n is the version number.

(*) the prompting character, indicates readiness to accept a command from the keyboard. At this point, any program linked for time-shared operation can be run.

5.6.3 Memory Allocation

The allocation of memory for an XMTSD system in which XMTSD is running as a background program is shown in Figure 5-3.

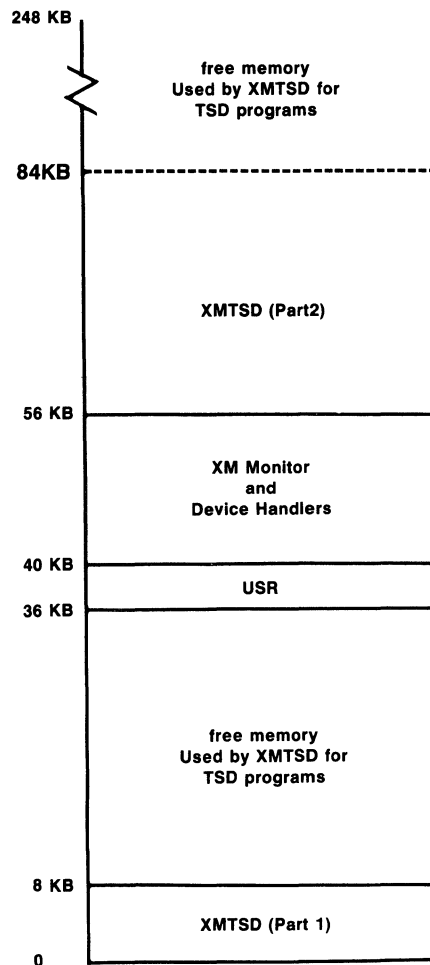


Figure 5-3 Memory Allocation for XMTSD as a Background Program

5.6.4 Applications

When XMTSD is run in the background (as contrasted with running in the foreground, see Section 5.7), the system can be thought of as running in a production mode. All the system resources, including lower memory and console terminal, are available for the day-to-day needs of application programs under the control of the run-time system.

5.7 XMTSD CHARACTERISTICS (FOREGROUND OPERATION)

The use of virtual overlays supported in RT-11 V4 (and later) makes it possible to run a Save image file using virtual overlays as either a background or as a foreground job. XMTSD is the only time-shared run-time system that uses virtual overlays. Therefore, only XMTSD can be run either as a foreground or as a background program.

If you choose to run XMTSD as a foreground job, the time-shared programs are restricted to running in the extended memory region. This frees a part of lower memory for use by other programs. These other programs could be Save image programs such as PIP, DUP, DIR, DICOMP, LINK, or SUD programs.

When XMTSD is running as a foreground program, the system can be thought of as running in a concurrent development mode. It is possible for XMTSD to communicate with a special program (LISTNR.SAV, supplied with CTS-300) running in the background. XMTSD has, as part of its system, a queuing routine for directing requests to this special background program. Sections 5.7.4 through 5.7.6 explain the communication process.

5.7.1 System Requirements

XMTSD requires that the XM monitor be selected during RT-11 system generation (SYSGEN). For foreground operation and communication with the background, implicit job startup must be selected during CTSGEN.

The new compiler is supplied in two forms. SJDBL.SAV operates with the SJ monitor and XMDBL.SAV operates with the XM monitor. The RT-11 DCL commands COMPILE/DIBOL or DIBOL require the compiler to be named DICOMP. Rename (via COPY) the desired compiler to DICOMP.SAV.

NOTE

Use XMDBL.SAV for concurrent programming.
(See below.)

Concurrent Development

With Version 8, concurrent development requires that an additional step be taken prior to running XMTSD. A new utility called SETVM writes and executes a command file that prepares the system for the compiler to be run as a virtual job using both high and low memory. Normally XMTSD assumes control of all upper memory. SETVM uses the virtual memory handler to set aside a portion of upper memory prior to running XMTSD. The amount set aside is calculated on the total memory available on your particular system. This memory set aside is, of course, not available for applications. The procedure is shown on the next page.

```

.REMOVE VM          ;
.R SETVM           ; Run the utility.
.                  ;
.                  ;
.                  ;
.FRUN XMTSD.SAV    ; Run XMTSD.
.                  ;
.                  ;
.                  ;
*CTRL/B           ; From the background terminal
B>                ;
.REMOVE VM        ; remove the virtual memory handler.
.R LISTNR         ; Set up foreground/background
.                ; communications and
.                ;
.                ;
.CTRL/F           ; return to the
F>                ;
*SUBMIT etc       ; foreground for concurrent
                  ; development.

```

Every time XMTSD is terminated (RTEXTIT), the system must again be set up as above.

NOTE

You must select the virtual memory handler during SYSGEN if you intend to do concurrent development.

Be aware that following the running of SETVM, which sets the base address for the virtual memory handler, any subsequent system boot will cause automatic installation of the virtual memory handler at that base address. Regardless of whether or not you want to do concurrent development, this will be true if you chose the handler during SYSGEN. XMTSD will then be limited in its use of memory. The easiest way to overcome this is to include a REM VM line in your startup command file.

5.7.2 Running XMTSD in the Foreground

Once the XMTSD program has been properly built via CTSGEN, time sharing can begin. Assume you have four terminals, including the console terminal, and that four terminals were selected in both the RT-11 system generation (SYSGEN) and the CTS-300 CTSGEN. It is recommended that you use the following procedure to execute the time-sharing program and the background listener program.

Load and execute the time-sharing program (XMTSD run-time system) by using the foreground RUN command in the form:

```
.FRUN filnam.SAV
```

where:

filnam is the name and extension of the XMTSD run-time system as specified in answer to the last question in CTSGEN (see Chapter 6). Regardless of the file name you choose for your XMTSD run-time system, XMTSD will always identify itself as XM-TSD.

- The extension .SAV must be specified.
- The program must be run from the system device.

Once started, the XMTSD run-time system identifies itself by displaying the following message at each of the terminals that it recognizes (in this case, terminals 1, 2, and 3):

```
XM-TSD VERSION RT11 XMT V0n.000
*
```

where:

n is the version number.

(*) the prompting character, indicates readiness to accept a command from the keyboard. At this point, you may set terminal characteristics but do not run any program unless you do not intend to run the listener.

If you want XMTSD to have access to the background, run the listener program at terminal 0, the background console. The command and response are:

```
.R LISTNR
CTS300 V8 LISTNR
```

Enter the following to return to the foreground to run time-sharing programs from the console (terminal 0):

```
.CTRL/F
```

5.7.3 Memory Allocation

The allocation of memory for an XMTSD run-time system in which XMTSD is run as a foreground program is shown in Figure 5-4.

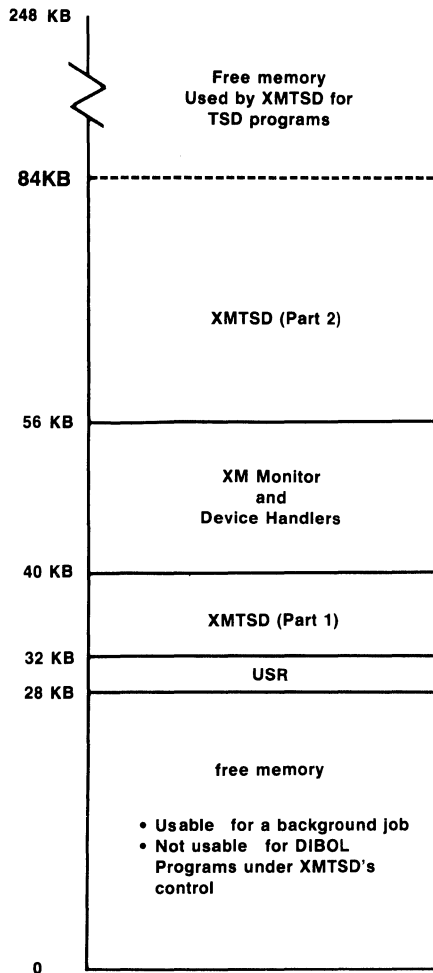


Figure 5-4 Memory Allocation for XMTSD as a Foreground Program

5.7.4 Foreground Queue Program

Foreground communication with the background is initiated via a special series of commands. It is important to note that these commands are, in reality, TSD programs constructed to implement the communications capability. The function of each is covered in detail in Section 5.7.6. A time-shared queue manager program, BGMAN.TSD, a part of XMTSD, is automatically invoked whenever one of these commands is issued. If BGMAN.TSD is not already running, it is automatically started up in a detached state to respond to the command request (hence the need for implicit job startup). The request is queued (16 requests maximum) by BGMAN.TSD and if the request requires it, the information is sent to LISTNR.SAV (see Section 5.7.5).

BGMAN.TSD has, as part of its code, a timing facility to allocate processor time between the foreground and the background. When BGMAN.TSD is started up, it automatically assigns a value of one to this timer. A value of one allocates 1/60th (or 1/50th - depending on the system clock) of a second to background operation for each XMTSD scheduler cycle. This allocation can be changed when the job is submitted to BGMAN (see the SUBMIT command, Section 5.7.6.1). The value is returned to one once the submitted request has been carried out. The advantage of increasing this value is to allocate a greater proportion of time to the submitted background job. This allows the job to execute faster; however, foreground programs will slow down proportionately.

If BGMAN.TSD has no pending requests in its queue, it will terminate itself. It is started (implicitly) later if there is a need for it to be involved.

5.7.5 Background Listener Program

The background program supplied by CTS-300 to receive the requests (indirect files) from the foreground is called LISTNR.SAV. This program is run at the console terminal by the user once XMTSD is running in the foreground. The command and response are:

```
.R LISTNR
CTS300 V8 LISTNR
```

When the indirect file from the foreground is sent to LISTNR in the background, LISTNR builds a one-block indirect file called SYSTEM.BKG. which contains:

```
@file.ext          ;USER PASSED INDIRECT FILE SPECIFICATION
                   ;RECONSTRUCTED FOR THE RT-11 SYSTEM
.R LISTNR          ;RESTART LISTNR
```

LISTNR submits the user request to RT-11 by chaining to SYSTEM.BKG. After the commands within the indirect file are processed, LISTNR is again started up, displays the version message, and informs XMTSD (BGMAN.TSD) that the job is done and that it is ready to receive the next command with its associated instructions in the form of another indirect file.

5.7.6 Foreground / Background Communications Commands

The special communication commands described in this section provide the interface between the keyboard and the foreground queue manager (BGMAN). The response to each command by both the foreground queue manager and LISTNR in the background is illustrated in Figure 5-5. Reference to this figure may help clarify the commands which follow.

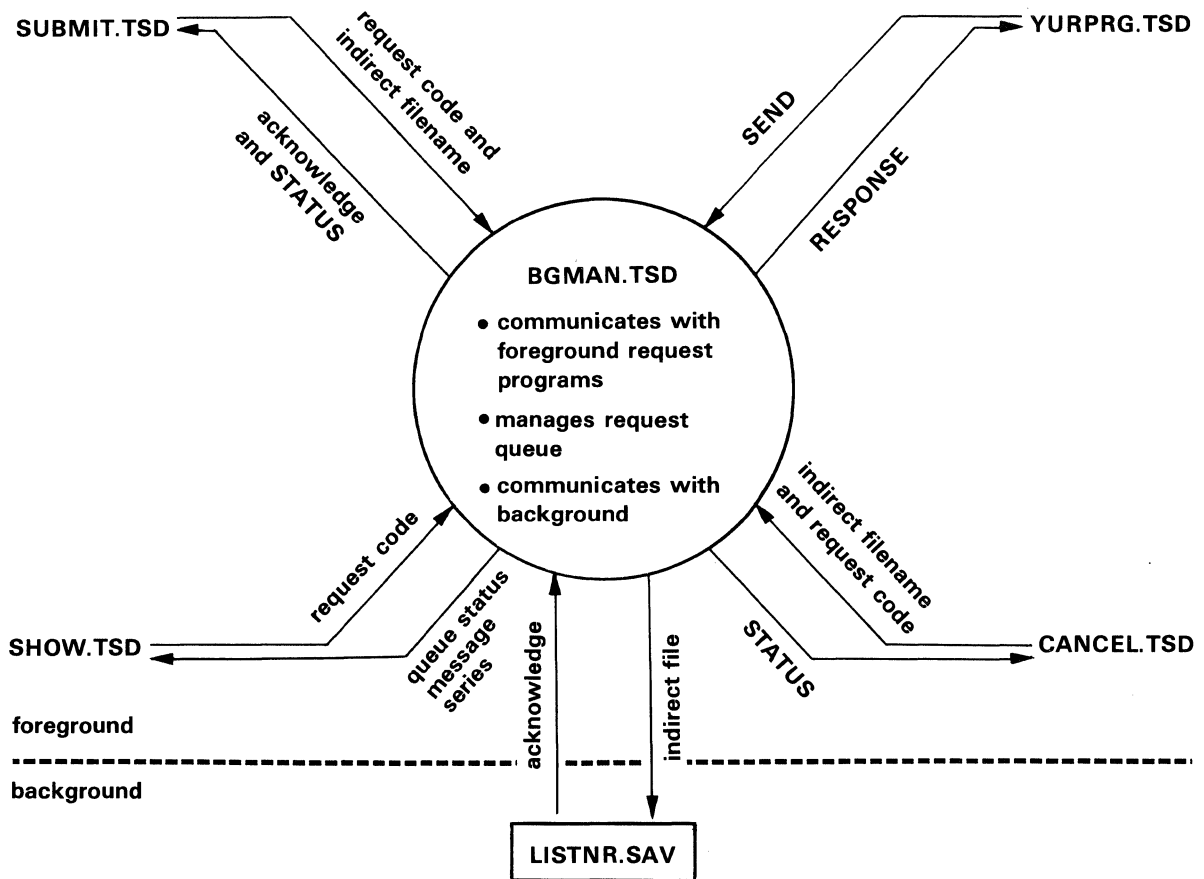


Figure 5-5 BGMAN

5.7.6.1 The SUBMIT Communication Command — This command is used to send an indirect file to LISTNR in the background. The indirect file previously constructed by the user is typically a list of instructions to be performed in the background. After receiving the indirect file name, the command-processing program (SUBMIT.TSD) issues a SEND to BGMAN.TSD. SUBMIT receives a response indicating the status of the request and passes it to the user.

The SUBMIT command has the form:

```
*SUBMIT filespec1...,filespecn[/n]
```

where:

filespec1...,filespecn

are up to nine file specifications of the indirect files to be passed to RT-11 in the background.

- If the file specification is omitted, you are prompted for input.

- Asterisk wildcards can be used for file name or extension.
- If no device is specified, the default device DK: is assumed.
- If no extension is specified, the default extension of .COM is assumed.

/n is an optional number specified by the user to allocate the time available for background processing. It allocates 1/60th (or 1/50th, depending on system clock) of a second to background operation for each XMTSD scheduler cycle. The usable range is from 1 to an upper limit determined by the particular application parameters. Two digits are allowed but anything greater than a single number usually has an unacceptable effect on the foreground operation. The default value of n is 3.

SUBMIT returns a message indicating the status of your request:

Assume that jobs A.COM and C.COM, requested from terminal 1, are already in queue with the default time allocation of 3 and that a submission request is made for a job B.COM with a time allocation of 5 from terminal 2 as follows:

```
*SUBMIT B.COM/5
```

The response is:

```
BACKGROUND QUEUE STATUS
```

```
ENTRIES: 3
```

```
CURRENT JOB: DK:X.COM
```

TERMINAL	FILENAME	TIME ALLOC
1	DK:A.COM	3
1	DK:C.COM	3
2	DK:B.COM	5

```
*** END ***
```

The current job listed above is the file being processed at the time the command was issued.

5.7.6.2 The SHOW Communication Command — The SHOW command allows you to determine the status of your request at some time after it was submitted and to show what other files are in the queue. SHOW does not communicate with the background. It simply receives a response from BGMAN.TSD regarding the contents of the queue file.

The command is entered in response to XMTSD's asterisk prompt:

```
*SHOW
```

The response is identical to the format illustrated for the SUBMIT command.

The SHOW command can be used to determine if LISTNR is running. If it is not running, an error message appears informing you of this fact.

5.7.6.3 The CANCEL Communication Command — The CANCEL command is issued whenever you want to retract a request while it is still in the BGMAN processing queue.

The CANCEL command has the form:

```
*CANCEL filespec1...,filespecn
```

where:

```
filespec1...,filespecn
```

are up to nine file specifications of files to be canceled. They are in the form dev:filnam.ext.

- If the file specification is omitted, you are prompted for input.
- Asterisk wildcards can be used for file name or extension.
- If no device is specified, the default device DK: is assumed.
- If no extension is specified, the default extension of .COM is assumed.

CANCEL returns the status of the queue thereby showing the file deleted. For example, a CANCEL request sequence for job A.COM (assuming A.COM, B.COM, and C.COM are in queue), would be:

```
*CANCEL A.COM
```

The response is:

```
BACKGROUND QUEUE STATUS
```

```
ENTRIES: 2
```

```
CURRENT JOB: DK:X.COM
```

TERMINAL	FILENAME	TIME ALLOC
1	DK:C.COM	3
2	DK:B.COM	5

```
*** END ***
```

The current job listed above is the file being processed at the time the command was issued.

5.7.7 Applications

When XMTSD is run in the foreground, the system can be thought of as running in what is primarily a development mode, because in this mode many activities normally associated with development are efficiently handled. Among these activities are a batch mode and remote patching. Possible operation as a production system is also discussed here.

Batch Mode

The entire process of application program development can be undertaken by several users of a time-shared DIBOL system. Any user at a foreground terminal can edit and debug under XMTSD. The DIBOL editor DKED.TSD can be used to create and edit source files. These files can be DIBOL programs or indirect files consisting of commands to be sent to the background. Furthermore, by communicating with the background program, LISTNR, programs can be compiled and linked.

Note that when compiling programs in this mode you must use the /G option to create an error log file which will contain possible errors. See Chapter 4 for more information on this option.

Remote Patching

Since it is possible for a remote terminal to communicate with the foreground program XMTSD and, through it, with the background, any remote terminal can be used to perform system maintenance, including patching.

Production Mode

While running XMTSD in the background is considered the conventional production mode (see Section 5.6), it is possible that running XMTSD in the foreground could be useful in some production environments. This would be an application that could make use of the background processing capability.

In such a situation, you would probably want all terminals, including the console terminal, to be available for production use. To accomplish this you must do the following:

- Suppress display of the LISTNR identifier. This is done by running RT-11 SIPP.SAV and setting location 1000 in LISTNR.SAV to a one.
- Set TT:QUIET for the console terminal.
- Ensure that the background operation you are performing does not output to the terminal.
- Be sure not to enter a CTRL/B which would attach the terminal to the background.

5.8 SUMMARY OF XMTSD CAPABILITIES

XMTSD capabilities are illustrated in Figure 5-6 for operation either in the background or in the foreground environment. When run in the foreground, the capabilities depend on whether LISTNR is running.

XMTSD USE	Capabilities				
	Production only	Production + 1 developer	Production + multi-developer	Production + remote diag. & maint.	Development only
Background	X				
Foreground w/o listener		X			
Foreground with listener	X		X	X	X

Figure 5-6 XMTSD Capabilities

Figure 5-6 shows that if you run XMTSD in the foreground with LISTNR in the background, you have the greatest choice of operating modes and system capabilities. However, if you are interested in production work only, running XMTSD in the background is the logical choice.

5.9 TERMINATING TIME-SHARED OPERATION (RTEXT)

5.9.1 Running RTEXT

The RTEXT program is used to terminate the operation of the time-shared system and the programs operating under its control. The procedure is the same for both TSD and for XMTSD when run in the background. For XMTSD in the foreground see Section 5.9.3. For TSD and XMTSD (background) the following command is entered while the time-shared system is operating:

```
*RTEXT
```

If no jobs are running, control returns immediately to the RT-11 monitor. If other jobs are running, the response is:

```
DO YOU WISH AN ABSOLUTE SHUTDOWN? (Y-N)
```

The default answer of Y causes any job currently running to be unconditionally terminated with unpredictable results for that job. If you answer N (or CR) RTEXT will display the current status of jobs every six seconds until such time as all jobs are completed. The display format is the same as with the J option in the STATUS utility (see Chapter 13). Control returns to the RT-11 monitor when all the jobs are completed. You can cancel the current status display by entering a CTRL/C at any time. Cancellation allows you to run RTEXT again to select the absolute shutdown or to simply run another TSD applications program.

5.9.2 Chaining to RTEXT

RTEXT may be automated by having a preceding program chain to it. The operation is then essentially the same as the situation where RTEXT is run manually and, if XMTSD is running in the foreground, the appropriate parts of the procedure shown in Section 5.9.3 are followed. The program line would be:

```
STOP 'RTEXT.TSD'
```

Optionally, a SEND statement can be used to specify the response to the absolute shutdown question and to specify a program to be executed following RTEXT. The SEND statement format to do this for a program running under the XMTSD system program (except XMTSD in the foreground) is:

```
SEND ('filnam ch', 'RTEXT.TSD')
```

where:

- | | |
|--------|--|
| filnam | is the name of a file to be executed once XMTSD is terminated and could be a save file or an indirect file. |
| ch | is an optional alpha character separated from the file name by a space. This character (Y or N) is the response to the absolute shutdown question. |

- An absolute shutdown occurs if the character is absent.
- If the program preceding the running of RTEXT is detached, the shutdown is always absolute regardless of the presence of an N alpha character.

5.9.3 RTEXT for XMTSD in the Foreground

When XMTSD is operating in the foreground, the selection of RTEXT is somewhat more involved because of possible interaction between foreground and background at the time RTEXT is run. The following is an example of the steps that may be required:

To start time-sharing XMTSD in the foreground:

```
.FRUN XMTSD.SAV
```

at the background terminal:

```
.R LISTNR
```

To terminate LISTNR running in the background, enter:

```
.CTRL/C
```

To terminate XMTSD which is running in the foreground, enter:

```
*RTEXT  
DO YOU WISH AN ABSOLUTE SHUTDOWN? (Y-N) (N) <CR>
```

The status of the current jobs running is displayed.

When all the jobs are complete and XMTSD terminates, at the console terminal, enter:

```
.UN F
```

This unloads the foreground job which is XMTSD.

CHAPTER 6

SYSTEM DEVELOPMENT

6.1 INTRODUCTION

A CTS-300 System is comprised of two major software packages: RT-11 and CTS-300. Both are furnished in a form that has the potential for many capabilities but which is probably not immediately or directly usable by any CTS-300 user. Each software package should be tailored to match the hardware and to provide the specific software capabilities needed by a given user. In this manual this process is called system development. Both RT-11 and CTS-300 have their own procedures to facilitate such development.

6.1.1 System Generation Programs

RT-11 software is adapted to the hardware and user requirements using the RT-11 system generation program RT-11 SYSGEN. The SYSGEN contains an option that allows CTS-300 users to make choices appropriate to their system. This option and these choices are explained in this chapter. CTS-300 software (both single-user and time-shared) is adapted to the user's requirements with a program called CTSGEN. CTSGEN is described in detail in this chapter.

6.1.2 Chapter Organization

The remainder of this chapter is comprised of two major sections. The first, Section 6.2, SYSGEN, describes the RT-11 system generation program as it applies to CTS-300. The second, Section 6.3, CTSGEN, provides detailed information on the CTSGEN program. This includes actual CTSGEN dialog and information to help you make choices.

6.1.3 CTSGEN Errors

CTSGEN checks your responses to verify that they are within the range of permissible answers. If an error is detected, you are advised that your answer exceeds the possibilities and you are prompted with the question again. CTSGEN does not detect logical errors (such as specifying more terminals than exist). However, in the assembling process after CTSGEN execution, MACRO errors can occur. If a message indicating that a file was not found appears during the assembling or linking process, one of the files specified in Section 6.3.1.2 is not present on the system device.

If you continue to receive errors, consult you DIGITAL software specialist.

NOTE

See the *CTS-300 Version 8 Release Notes and Installation Guide* for the latest information regarding any CTSGEN cautions or changes.

6.2 SYSGEN

The first step in developing a working system is to build an appropriate RT-11 Operating System. You must run RT-11 SYSGEN to select the RT-11 monitor services and device handlers you require. Since the RT-11 dialog is long and repetitive, an optional, shorter list is provided for CTS-300 users. This shorter version is the default option.

NOTE

- You must select the virtual memory handler during SYSGEN if you intend to do concurrent development.
- Software products that make use of CTS-300 (layered products such as DECtype) may impact SYSGEN and CTSGEN responses. In general, see the installation documentation for such products for special requirements.

The questions that have been rephrased are:

Question	Change
First question following:	Do you want an introduction to system generation (N)?
now reads:	Do you want the CTS-300 dialog (Y)?
Q3 reads:	Do you want the extended memory (XM) monitor (Y)?
Q9 reads:	Do you want to use TSD with this monitor (Y)?
Q14:	Note that CTS-300 does not make use of this feature (.FETCH request).
Q45 reads:	Do you want serial line printer support (Y)?
Q46 reads:	How many (serial & parallel) line printers (1)?
Q47 reads:	Is the first [second, etc] line printer parallel (Y)?

The answers automatically provided with the CTS-300 SYSGEN are:

	Question	Answer
Q4	SJ timer support	Yes
Q6	Error message to replace system halt upon receipt of a system I/O error for the single-job monitor	Yes
Q8	.SPCPS request	Yes
Q10	Asynchronous terminal status	Yes
Q25	KW11-P clock as the system clock	No
Q27	Floating point support	No
NA	Retention of work files and .OBJ files (Not a numbered question. Appears at the end of the session.)	No

If you want to change any of the automatic selections, answer NO to the CTS-300 dialog question. By responding NO, you will have chosen the full RT-11 SYSGEN, and you will then be asked the first question in the RT-11 SYSGEN. If you choose RT-11 SYSGEN, and you plan to run a time-shared DIBOL program, you must answer YES to .SPCPS request (Q8), choose multiterminal support (Q9), and answer YES to asynchronous terminal status (Q10).

You should refer to the *RT-11 Installation and System Generation Guide* before you start, because you must know what your requirements will be before you run SYSGEN. It is important to remember your SYSGEN responses dealing with terminals, because they effect your answers in CTSGEN.

When you have completed your RT-11 SYSGEN, you will be ready to run CTSGEN.

6.3 CTSGEN

The CTSGEN program is the CTS-300 Operating System Program generator. It is an interactive program that is used to select the parameters associated with a CTS-300 system. It can create a run-time code interpreter for a Single-User DIBOL (SUD) program, or it can create both a code interpreter and run-time system executive for a time-shared system (TSD or XMTSD). As with the RT-11 SYSGEN it is wise to plan ahead so you will know exactly what your requirements are before building the CTS-300 system. It may help if you read Chapter 5 in this manual, before initiating any CTSGEN activity.

6.3.1 Characteristics

6.3.1.1 Choices — CTSGEN allows you to build either a Single-User DIBOL run-time system program or a Time-Shared DIBOL run-time system program.

Single-User System

If you are building a SUD system, the choices are limited to the selection of two system services and location of the error message file. You can select:

- support for ISAM files
- support for DDT
- placement of the error message file

Time-Shared System

If you are building a time-shared run-time system, there are two basic types of support services you can select:

Run-time system services:

- number of programs
- number of messages stored
- number of channels to be used
- number of files open at any one time
- support for ISAM files
- support for the XM monitor (and residency of USR)
- support for DDT
- support for implicit or forced-job startup
- auto job startup upon conclusion of the time-shared run-time system load
- support for fatal error handling
- placement of the error message file

Peripheral device support:

- local or remote terminal use
- DL11 or DZ11 terminal interface
- mechanical operating characteristics of each terminal
- total number of different peripheral devices

6.3.1.2 Preliminary Requirements — As you did with SYSGEN, you must plan ahead for your CTSGEN session. Acquaint yourself with the flow of CTSGEN by using the chart, Figure 6-1 in Section 6.3.2, and by reading the dialog and responses in the following sections for the system you intend to build. Remember that your responses in SYSGEN dealing with terminals effect your answers in CTSGEN. Record the number of terminals by interface and local/remote status.

Single-User System

The following modules are necessary for building a single-user run-time system. They must be on the system disk.

CTSGEN.SAV	IO.OBJ	LINK.SAV
DDT.OBJ	ISAM.OBJ	MATH.OBJ
DDTX.OBJ	ISAMX.OBJ	SDIRT.OBJ
ERRNM.MAC	JOB.OBJ	SUD.RTS
ERRNM.OBJ		

Time-Shared System

Use the lists in Section 6.3.1.1 and note the peripheral devices on your system and the run-time system services you require for your application.

The following modules are necessary for building a time-shared run-time system (TSD or XMTSD). They must be on the system disk.

CTSGEN.SAV	DERROR.OBJ	KDDT.OBJ
LINK.SAV	DISAM.OBJ	KDDTX.OBJ
MACRO.SAV	DISAMX.OBJ	KDIRT.OBJ
SUD.RTS	DMATH.OBJ	KEROR2.OBJ
SYSMAC.SML	DMESS.OBJ	KERROR.OBJ
TD.MAC	DJOB.OBJ	KISAM.OBJ
TSDTBL.MAC	DIO.OBJ	KISAMX.OBJ
DEFS.MAC	DTO.OBJ	KMATH.OBJ
TSDDFN.MAC	DTOINI.OBJ	KDMESS.OBJ
QC.MAC	ERRNM.MAC	KJOB.OBJ
ST.MAC	ERRNM.OBJ	KDIO.OBJ
DATX.OBJ	FRUNIT.OBJ	KDIO.OBJ
DDDT.OBJ	FRUNXX.OBJ	KDIO.OBJ
DDDTX.OBJ	TTY.OBJ	KTOINI.OBJ
DDIRT.OBJ	KCORE.OBJ	KFRUN.OBJ
DEROR.OBJ	KDATX.OBJ	KFRUNX.OBJ
		KTTY.OBJ

6.3.1.3 Question Types — Each CTSGEN question has a default answer. That default is shown immediately after the question and is placed within parentheses. After each question is displayed, you respond by typing the desired entry or by typing a carriage return (<CR>) to select the default.

There are two kinds of questions in CTSGEN:

- Questions that require a Y or an N response.
- Questions that require a numeric response.

Unlike SYSGEN, there is no need to specify numbers in octal. Questions that require a numeric response accept only decimal numbers. The default decimal value is contained within parentheses immediately following the question.

If you need to change your response to a previous question, you may return to any previous question by entering Q followed by the question number.

Take special care when responding to questions that require other than Y, N, or a default value.

6.3.2 CTSGEN Dialog

This section describes all text, questions, and responses displayed during the dialog. However, during any CTSGEN session some of these questions and comments will not appear since their occurrence depends on answers to previous questions.

CTSGEN dialog appears as a series of questions with default answers in parentheses. If you enter an invalid response (or a question mark) in response to a CTSGEN question, the question (and default) is repeated along with additional comments. The comments can guide you through a CTSGEN session, and if you are not experienced with CTSGEN, will make you aware of the implications of your answers and help you avoid inaccurate responses.

Figure 6-1 illustrates the flow of questions in CTSGEN.

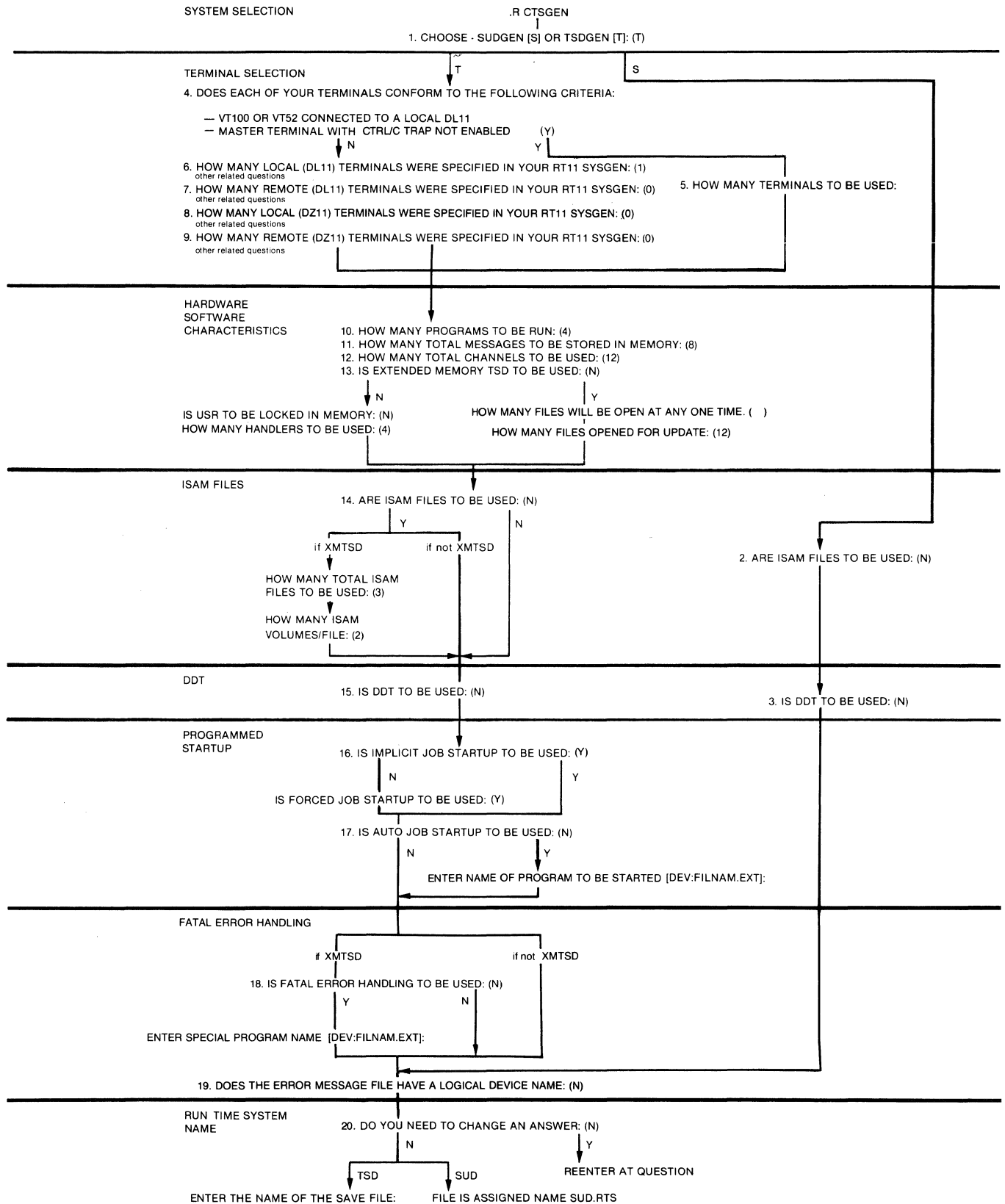


Figure 6-1 CTS-300 Operating System Generator (CTSGEN)

The following is the actual CTSGEN dialog. It is broken into sections in this manual to identify more clearly the activity taking place. The displayed dialog is shown in upper case only. Explanatory material is shown in both uppercase and lowercase.

CTSGEN is executed with the following command and responds as shown:

```
.R CTSGEN
```

```
CTSGEN Vnn-nn
```

```
EACH OF THE FOLLOWING QUESTIONS IS FOLLOWED BY A DEFAULT RESPONSE IN PARENTHESIS. THIS RESPONSE WILL BE USED IF A <CR> IS TYPED IN ANSWER TO THE QUESTION. IF A QUESTION MARK OR AN ILLEGAL RESPONSE IS TYPED, FURTHER INFORMATION CONCERNING THE CURRENT QUESTION WILL BE PRINTED AT THE TERMINAL. YOU MAY RETURN TO ANY OF THE QUESTIONS THAT ARE MARKED WITH A LINE NUMBER. SIMPLY TYPE THE LETTER Q AND THEN FOLLOW IT BY THE LINE NUMBER. (i.e., Q1, Q3, Q10)
```

6.3.2.1 Single-User or Time-Shared System — The first question in CTSGEN:

1. CHOOSE - SUDGEN [S] OR TSDGEN [T]: (T)

If you are going to build a time-shared system, respond with T (or <CR>). If you are going to build a single-user system, respond with S. If you answer T, Q4 will appear. See Section 6.3.2.3.

6.3.2.2 Single-User System — There are only two questions asked for a single-user system:

2. ARE ISAM FILES TO BE USED: (N)

Choosing ISAM costs approximately 4.2 K bytes.

Respond with N (or <CR>), if you do not plan to use ISAM files. Respond with Y if you do plan to use ISAM files. The second question:

3. IS DDT TO BE USED: (N)

Choosing DDT costs approximately 1100 bytes.

Respond with N (or <CR>) if you do not plan to use the DIBOL debugging program, DDT. Respond with Y if you do plan to debug. The next question:

19. DOES THE ERROR MESSAGE FILE HAVE A LOGICAL DEVICE NAME: (N)

The error message file can be placed on any device (physical or logical). Respond with the name of the device. The run-time system will thereafter look first for the error message file on this device. If the file is not found on the device, a search is made of DK: followed by a search of SY:. If an N response is made, the system looks for the file first on DK: followed by SY:.

This completes the choices for a single-user system. You are given the opportunity to change your answers:

20. DO YOU NEED TO CHANGE AN ANSWER: (N)

If you are satisfied that your answers reflect your hardware and software needs, respond with an N or (CR). If you want to change an answer, respond with a Y which prompts:

REENTER AT QUESTION:

Respond with a Q followed by the question number or simply with the question number. Questioning will begin again at the point selected.

With your N or (<CR>) answer to Q20, CTSGEN proceeds to automatically build a run-time code interpreter for Single-User DIBOL (SUD) programs. The name of the interpreter is chosen by CTSGEN to be SUD.RTS and is automatically stored on the system disk. A link map (SUD.MAP) is also generated which may be helpful if you have a problem with the interpreter.

The file SUD.RTS must always be on the system disk for your SUD system. You cannot run any Single-User DIBOL program, not even CTSGEN, without SUD.RTS. Every time you run CTSGEN, a new CTSGEN.COM file is generated. For this reason, you may want to rename the CTSGEN.COM file to SUDGEN.COM (for example) after this CTSGEN and before another.

Use the command string:

.RENAME CTSGEN.COM SUDGEN.COM

This allows you to recreate the same SUD.RTS selected by questions 2, 3, and 19 above. All you have to do is type @SUDGEN. This can be very helpful if you should accidentally destroy your SUD.RTS.

It is permissible, and often useful, to create several versions of SUD.RTS and store them for later use. Considering ISAM and DDT, there are four possibilities depending on choice of support or non support. The distributed version is without ISAM or DDT support. Note that other versions must be stored with names other than SUD.RTS, since SUD programs automatically look for SUD.RTS when the RUN command is issued. Whenever a particular version is needed, you simply rename it to SUD.RTS (after storing its predecessor under its storage name) and proceed to run your SUD programs.

6.3.2.3 Time-Shared System — The first question in CTSGEN related to a time-shared system:

4. DOES EACH OF YOUR TERMINALS CONFORM TO THE FOLLOWING CRITERIA:

- VT100 OR VT52 CONNECTED TO A LOCAL DL11
- MASTER TERMINAL WITH CTRL/C TRAP NOT ENABLED (Y)

Terminals are assigned to the DL11 (or other) interface during SYSGEN. Local terminals are assigned first for each interface.

A master terminal is a terminal from which you can initiate a time-shared program. This cannot be done at a slave terminal.

CTRL/C trap prevents a CTRL/C from terminating a time-shared program. Such abnormal termination could result in loss or corruption of data files.

If any one of the terminals does not meet the above criteria, you must respond N. You are then asked specific questions (starting with Q6 in the next section) concerning each of the terminal lines specified in SYSGEN.

If your answer is Y or <CR> :

5. HOW MANY TERMINALS TO BE USED: (2)

You may select up to the number chosen in SYSGEN. CTSGEN will permit up to 12. A limit of eight is recommended for systems in which terminal input will be heavy. A valid response prompts Q10 to be displayed; please see Section 6.3.2.9 to continue selection of the other system parameters.

6.3.2.4 Terminal Specification — Since your response to question 4 was N (your terminals do not meet the criteria stated in Q4), more information is needed. Terminal information to be specified in the following sections of this chapter (CTSGEN questions 6 through 9) depends on the following factors.

Your answers to SYSGEN set the limits for your answers in CTSGEN. SYSGEN used your answers to assign hardware interfaces to support the terminals. CTSGEN asks you which of these terminals you want your time-shared program to recognize and for the characteristics of these terminals.

CTSGEN terminal questions are presented in the same four categories as in SYSGEN: DL11 local, DL11 remote, DZ11 local, and DZ11 remote. In each category you are asked which terminals are to be used and which are to be unused. The total (used and unused) must equal the SYSGEN assignment for that category.

Additional information pertaining to these terminals is also asked in each of the four categories.

6.3.2.5 Local DL11 Terminals — This series of questions pertains to local terminals on the DL11 interface.

6. HOW MANY LOCAL (DL11) TERMINALS WERE SPECIFIED IN YOUR RT11 SYSGEN: (1)

Respond with the number of local DL11 lines that you specified during SYSGEN. You must specify at least one your console terminal.

Now you must enter the operating characteristics of each terminal. A series of five questions follows for each terminal identified in Q6.

TERMINAL n IS TERMINAL n TO BE USED: (Y)

Respond Y or <CR> if a terminal is attached to this line and you want to use it. From within a DIBOL program, terminal 1 would be referred to as terminal 0. An N response will either cause this question to be asked for the next terminal or else cause Q7 to be asked.

IS TERMINAL n A SCOPE: (Y)

Respond Y or <CR> if this terminal has a display (video) screen. Respond N if this terminal is a teleprinter (prints out hard copy).

IS TERMINAL n A SLAVE: (N)

Respond N or <CR> if this terminal is to be a master terminal. Respond Y if you do not want this terminal to be able to start jobs via keyboard command. A Y response means that jobs at this terminal must be started by forced-job startup. Remember, if this is the console terminal, it must be a master terminal.

IS CTRL/C TRAP TO BE USED: (N)

Respond N or <CR> if you want a CTRL/C to be recognized by the time-shared run-time system. Respond Y if you want a CTRL/C to be ignored by the time-shared run-time system. This ignored CTRL/C is not passed to the DIBOL program.

HOW MANY FILL CHARACTERS TO BE USED: (0)

Respond with a decimal number according to your terminal model as noted in the following chart. The chart notes the number of fill characters required for each terminal model set for the baud rate shown. This number provides a delay time to allow completion of the carriage return / line feed.

Terminal	Baud rate set for	Fill characters
VT100	9600	0
VT50H	9600	0
VT52	9600	0
LA36	300	0
LA30	300	10
	150	4
	110	2
VT05	2400	2
	1200	2
	600	1

The preceding five questions appear for each local DL11 terminal you specified in question Q6 until operating characteristics are detailed for all local DL11 terminals to be supported at run time.

Now consider DL11 terminals for remote use.

6.3.2.6 Remote DL11 Terminals — This series of questions pertains to remote terminals on the DL11 interface.

7. HOW MANY REMOTE (DL11) TERMINALS WERE SPECIFIED IN YOUR RT11 SYSGEN: (0)

Respond with the number of remote DL11 lines that you specified during SYSGEN.

Now you must enter the operating characteristics of each terminal. A series of five questions follows for each terminal identified in Q7.

TERMINAL n
IS TERMINAL n TO BE USED: (Y)

Respond Y or <CR> if a terminal is attached to this line and you want to use it. An N response will either cause this question to be asked for the next terminal or else cause Q8 to be asked.

IS TERMINAL n A SCOPE: (Y)

Respond Y or <CR> if this terminal has a display (video) screen. Respond N if this terminal is a teleprinter (prints out hard copy).

IS TERMINAL n A SLAVE: (N)

Respond N or <CR> if this terminal is to be a master terminal. Respond Y if you do not want this terminal to be able to start jobs via keyboard command. A Y response means that jobs at this terminal must be started by forced-job startup.

IS CTRL/C TRAP TO BE USED: (N)

Respond N or <CR> if you want a CTRL/C to be recognized by the time-shared run-time system. Respond Y if you want a CTRL/C to be ignored by the time-shared run-time system. This ignored CTRL/C is not passed to the DIBOL program.

HOW MANY FILL CHARACTERS TO BE USED: (0)

Respond with a decimal number according to your terminal model as noted in the following chart. The chart notes the number of fill characters required for the LA30 for the baud rates shown (other terminals do not require fill characters). This number provides a delay time to allow completion of the carriage return / line feed.

Terminal	Baud rate set for	Fill characters
LA30	300	10
	150	4
	110	2

The preceding five questions appear for each remote DL11 terminal you specified in Q7 until operating characteristics are detailed for all remote DL11 terminals to be supported at run time.

Now consider terminals that interface with DZ11 hardware.

6.3.2.7 Local DZ11 Terminals

This series of questions pertains to local terminals on the DZ11 interface.

8. HOW MANY LOCAL (DZ11) TERMINALS WERE SPECIFIED IN YOUR RT11 SYSGEN: (0)

Respond with the number of local DZ11 lines that you specified during SYSGEN.

Now you must enter the operating characteristics of each terminal. A series of six questions follows for each terminal identified in Q8.

TERMINAL n IS TERMINAL n TO BE USED: (Y)

Respond Y or <CR> if a terminal is attached to this line and you want to use it. An N response will either cause this question to be asked for the next terminal or else cause Q9 to be asked.

IS TERMINAL n A SCOPE: (Y)

Respond Y or <CR> if this terminal has a display (video) screen. Respond N if this terminal is a teleprinter (prints out hard copy).

IS TERMINAL n A SLAVE: (N)

Respond N or <CR> if this terminal is to be a master terminal. Respond Y if you do not want this terminal to be able to start jobs via keyboard command. A Y response means that jobs at this terminal must be started by forced-job startup.

IS CTRL/C TRAP TO BE USED: (N)

Respond N or <CR> if you want a CTRL/C to be recognized by the time-shared run-time system. Respond Y if you want a CTRL/C to be ignored by the time-shared run-time system. This ignored CTRL/C is not passed to the DIBOL program.

HOW MANY FILL CHARACTERS TO BE USED: (0)

Respond with a decimal number according to your terminal model as noted in the following chart. The chart notes the number of fill characters required for each terminal model set for the baud rate shown. This number provides a delay time to allow completion of the carriage return / line feed.

Terminal	Baud rate set for	Fill characters
VT100	9600	0
VT50H	9600	0
VT52	9600	0
LA36	300	0
LA30	300	10
	150	4
	110	2
VT05	2400	2
	1200	2
	600	1

The next question:

WHAT IS THE LOCAL TERMINAL BAUD RATE: (9600)

Respond with a decimal number according to your terminal model type and the speed that is appropriate for it. Baud rates are:

110,	150	300,	600,	1200,
1800,	2000,	2400,	3600,	4800,
7200,	9600.			

The preceding six questions appear for each local DZ11 terminal you specified in Q8 until operating characteristics are detailed for all DZ11 terminals (local) to be supported at run time.

Now consider your DZ11 terminals for remote use.

6.3.2.8 Remote DZ11 Terminals

This series of questions pertains to remote terminals on the DZ11 interface.

9. HOW MANY REMOTE (DZ11) TERMINALS
WERE SPECIFIED IN YOUR RT11 SYSGEN: (0)

Respond with the number of remote DZ11 lines that you specified during SYSGEN.

Now you must enter the operating characteristics of each terminal. A series of six questions follows for each terminal identified in Q9.

TERMINAL n
IS TERMINAL n TO BE USED: (Y)

Respond Y or <CR> if a terminal is attached to this line and you want to use it. An N response will either cause this question to be asked for the next terminal or else cause Q10 to be asked.

IS TERMINAL n A SCOPE: (Y)

Respond Y or <CR> if this terminal has a display (video) screen. Respond N if this terminal is a teleprinter (prints out hard copy).

IS TERMINAL n A SLAVE: (N)

Respond N or <CR> if this terminal is to be a master terminal. Respond Y if you do not want this terminal to be able to start jobs via keyboard command. A Y response means that jobs at this terminal must be started by forced-job startup.

IS CTRL/C TRAP TO BE USED: (N)

Respond N or <CR> if you want a CTRL/C to be recognized by the time-shared run-time system. Respond Y if you want a CTRL/C to be ignored by the time-shared run-time system. This ignored CTRL/C is not passed to the DIBOL program.

HOW MANY FILL CHARACTERS TO BE USED: (0)

Respond with a decimal number according to your terminal model as noted in the following chart. The chart notes the number of fill characters required for the LA30 for the baud rates shown (other terminals do not require fill characters). This number provides a delay time to allow completion of the carriage return / line feed.

Terminal	Baud rate set for	Fill characters
LA30	300	10
	150	4
	110	2

The next question:

WHAT IS THE REMOTE TERMINAL BAUD RATE: (300)

Respond with a decimal number according to your terminal model type and the speed that is appropriate for it. The baud rate must be the same as that chosen in the SYSGEN question for DZ11 support. Baud rates are: 110, 150, and 300.

The preceding six questions appear for each remote DZ11 terminal you specified in Q9 until operating characteristics are detailed for all remote DZ11 terminals to be supported at run time.

You have completed the entry of details for all your terminals. The system hardware/software configuration questions appear next beginning with Q10.

6.3.2.9 System Hardware/Software Configuration — This section contains system-related questions. The answers are used to establish the capabilities and characteristics of the system you are generating.

NOTE

Many of the system-related questions allow a choice of a range of values for an answer. Unnecessary selection of maximum values can degrade system performance or even result in an XMTSD system that is too large to run.

The first of the system-related questions:

10. HOW MANY PROGRAMS TO BE RUN: (4)

The cost is approximately 60 bytes per program selected.

Respond with the decimal number of programs or jobs that you plan to run at any one time. The range is from 1 through 16.

11. HOW MANY TOTAL MESSAGES TO BE STORED IN MEMORY: (8)

The cost is approximately 14 bytes per message.

Respond with the decimal number of messages that can be stored in memory at any one time. The range is from 1 through 50. This number depends on your programming needs. Consider which of your programs use SEND or RECEIVE statements and how many messages are generated (LPTSPL.TSD and BGMAN.TSD use this facility).

12. HOW MANY TOTAL CHANNELS TO BE USED: (12)

The cost is approximately 31 bytes per channel selected.

Respond with a decimal number. The supported range is from 1 through 70. This refers to I/O channels (referenced by the DIBOL OPEN statement) that are needed for devices or files at any one time across all programs. In addition, your number sets the limit for questions prompted in Q13 (files opened for update) and in Q14 (total ISAM files).

If you respond with more than 70 channels, a statement of supported limits is displayed followed by the following message:

```
EXCEEDING THIS LIMIT >>> MAY <<< RESULT IN TSD BEING TOO BIG
TO RUN. SELECTING MORE THAN 70 CHANNELS IS >>>NOT <<<
GUARANTEED TO WORK. SUCCESSFUL SELECTION OF MORE THAN
70 CHANNELS DOES >>> NOT <<< ASSURE SUCCESS IN THE FUTURE.
```

DO YOU WISH TO SELECT MORE THAN 70 CHANNELS?

A Y response confirms your intention to exceed the recommended limit of 70 channels. Your choice cannot be greater than 99 channels under any circumstances, and any selection over 70 can result in a system that is too large to run. An N response causes Q12 to be repeated. Your final response to Q12 prompts the next question:

13. IS EXTENDED MEMORY TSD TO BE USED: (N)

Respond Y if you intend to run XMTSD. An answer of Y automatically implies that the User Service Routine (USR) will be locked in memory and the next two questions, the USR and device questions, do not appear. Respond N or <CR> if you do not need extended memory support for DIBOL. If you answer N or <CR> the next two questions are asked:

IS USR TO BE LOCKED IN MEMORY: (N)

Choosing USR to be locked in memory costs approximately 4.2 K bytes.

Respond N or <CR> if you want the USR to be swapped out during program execution. You will want the USR to be swapped out if memory space is too small to permit program execution. Respond with a Y if you want the USR to remain in memory permanently. With the USR locked in memory, files OPEN and CLOSE much faster.

HOW MANY HANDLERS TO BE USED: (4)

The cost is approximately 10 bytes per device selected.

Respond with the decimal number of handlers. The range is from 1 through 10. This number indicates categories of devices with the exception of printers. Each line printer requires a separate device handler.

An answer of Y to Q13 will prompt the next two questions; otherwise Q14 is asked:

HOW MANY FILES WILL BE OPEN AT ANY ONE TIME: (*)

(*) The default is the number of channels minus the number of terminals.

The cost (in bytes) is approximately 24 bytes per file selected.

Respond with the total number of files to be open in any mode across all jobs at any one time. The supported limit cannot exceed the total number of channels to be used. This question refers to the number of different files to be open at any one time. If two or more programs have the same file open this counts as one file. The mode can be the same or different. For example:

Program A has three files open:

FILE1.DDF	I mode
FILE2.DDF	U mode
FILE3.DDF	O mode

Program B also has three files open:

FILE1.DDF	U mode
FILE2.DDF	U mode
FILE4.DDF	I mode

If these two programs are running at the same time and if all the files are open, the correct answer to the question would be 4.

NOTE

The answer to the question is not affected by the number of terminals in use.

The next question:

HOW MANY FILES WILL BE OPENED FOR UPDATE: (12)

The cost (in bytes) is approximately:

$4 \times [\text{number of jobs}] \times [\text{number of files opened for update}]$.

Respond with a decimal number. This is the maximum number of different files opened in U-mode or SU-mode at any one time across all jobs. The supported range is from 0 through 25 or the answer to the previous question, whichever is the lesser.

If the answer to Q12 was greater than 25 and you respond with more than 25 files here, a statement of supported limits is displayed followed by the following message:

EXCEEDING THIS LIMIT >>> MAY <<< RESULT IN TSD BEING TOO BIG TO RUN. SELECTING MORE THAN 25 UPDATE FILES IS >>> NOT <<< GUARANTEED TO WORK. SUCCESSFUL SELECTION OF MORE THAN 25 UPDATE FILES DOES >>> NOT <<< ASSURE SUCCESS IN THE FUTURE.

DO YOU WISH TO SELECT MORE THAN 25 UPDATE FILES?

A Y response confirms your intention to exceed the recommended limit of 25 files. Your choice cannot be greater than the answer to Q12 (total channels to be used) under any circumstances, and any selection over 25 can result in a system that is too large to run. An N response causes Q13 to be repeated. Your final response to Q13 prompts the next question:

14. ARE ISAM FILES TO BE USED: (N)

Choosing ISAM costs approximately 4.2 K bytes.

Respond N or <CR> if ISAM files are not to be accessed in this version of the time-shared run-time system. Respond Y if ISAM files are to be created or accessed in this version of the time-shared run-time system. If you answer Y now, and you have answered Y to Q13 concerning extended memory time-sharing, two additional questions appear; otherwise Q15 is the next question:

HOW MANY TOTAL ISAM FILES TO BE USED: (3)

The cost is a function of both this question and "HOW MANY ISAM VOLUMES/FILE" following; therefore, see the discussion for that question before answering this.

The total number of ISAM files is the maximum number of ISAM files to be open at any one time across all jobs. Respond with a decimal number. The range is from 1 up to the number you specified in answer to Q12 (up to a supported maximum of 25).

If you respond with more than 25 files, a statement of supported limits is displayed followed by the following message:

EXCEEDING THIS LIMIT >>> MAY <<< RESULT IN TSD BEING TOO BIG TO RUN. SELECTING MORE THAN 25 ISAM FILES IS >>> NOT <<< GUARANTEED TO WORK. SUCCESSFUL SELECTION OF MORE THAN 25 ISAM FILES DOES >>> NOT <<< ASSURE SUCCESS IN THE FUTURE.

DO YOU WISH TO SELECT MORE THAN 25 ISAM FILES?

A Y response confirms your intention to exceed the recommended limit of 25 files. Your choice cannot be greater than the answer to Q12 (total channels to be used) under any circumstances, and any selection over 25 can result in a system that is too large to run. A N response causes the question to be repeated. Your final responses to these questions prompt the next question:

HOW MANY ISAM VOLUMES/FILE: (2)

The cost (in bytes) is approximately:

the number of files X [20 + [10 X the number of volumes per file]].

Respond with a decimal number. The range is from 2 through 8; however, ISAM volumes per file times the number of ISAM files (previous question) must be less than or equal to the greater: 50 or two times the response to "HOW MANY TOTAL ISAM FILES TO BE USED." The ISAM file that occupies the greatest number of volumes will determine your response. If you have prompted CTSGEN for explanatory remarks with either of the two preceding questions, or if you have exceeded limits set for these questions, the questions will be asked again; otherwise, your response prompts:

15. IS DDT TO BE USED: (N)

The cost is approximately 1.2 K bytes.

Respond N or <CR> if you will not require the use of DDT in this time-shared run-time system. Respond Y if you require the use of DDT in this time-shared run-time system.

16. IS IMPLICIT JOB STARTUP TO BE USED: (Y)

Implicit job startup costs approximately 100 bytes.

Respond Y or <CR> if you want the capability of implicit job startup. This will allow a job to start in response to a SEND statement. Selecting implicit job startup automatically selects forced job startup, below (with its additional cost of 280 bytes); and that question is therefore not asked. Implicit job startup must be selected if you intend to run XMTSD as a foreground program. If you do not want the capability of implicit job startup, respond N.

If you answer N to the preceding question, you are asked:

IS FORCED JOB STARTUP TO BE USED: (Y)

Forced job startup costs approximately 280 bytes.

Respond Y or <CR> if you want the capability for forced job startup. This enables you to start a job on another terminal with an XCALL to RUNJB. Respond with N if you do not want to use this capability. You must respond with Y if the line printer spooler LPTSPL.TSD is to be used.

17. IS AUTO JOB STARTUP TO BE USED: (N)

Auto job startup costs approximately 30 bytes.

Respond N or <CR> if you do not want to specify a DIBOL program to be started after the time-shared run-time system is loaded. Respond with Y if you want to automatically start a DIBOL program upon completion of the time-shared run-time system load. If you answer Y, you are prompted for the name of the program to be automatically started:

ENTER NAME OF PROGRAM TO BE STARTED [DEV:FILNAM.EXT]:

If you have answered N to Q13 concerning extended memory time-sharing, the next question asked is Q19. If you answered Y to Q13 (yours is an extended memory time-sharing system) you are asked:

18. IS FATAL ERROR HANDLING TO BE USED: (N)

Respond N or CR if you do not want XMTSD to execute a special program in place of the application program if the application should encounter a fatal error. An XCALL to the FATAL external subroutine (see below) enables fatal error handling for the program making the call even if Q18 were answered N. If you respond Y you will be prompted for the name of the program to be executed in place of the application program:

ENTER SPECIAL PROGRAM NAME [DEV:FILNAM.EXT]:

When a fatal error occurs, the run-time system passes diagnostic error information to the program named above. Once fatal error handling is chosen, a name can be supplied with an XCALL to the FATAL external subroutine which overrides the one specified here. If the FATAL program cannot be found, full error traceback is reported in the same manner as if FATAL error handling had not been chosen. The next question:

19. DOES THE ERROR MESSAGE FILE HAVE A LOGICAL DEVICE NAME: (N)

The error message file can be placed on any device (physical or logical). Respond with the name of the device. The run-time system will thereafter look first for the error message file on this device. If the file is not found on the device, a search is made of DK: followed by a search of SY:. If an N response is made, the system looks for the file first on DK: followed by SY:.

You have now selected all the system parameters, and are asked if you want to change any of your answers.

20. DO YOU NEED TO CHANGE AN ANSWER: (N)

If you are satisfied that your answers reflect your hardware and software needs, respond with N or <CR>. If you want to change an answer, respond with Y which prompts:

REENTER AT QUESTION:

Respond with Q followed by the question number or simply with the question number. Questioning will begin again at the point selected.

6.3.2.10 Naming the Time-Shared Program — If you are satisfied with all your answers, the final question is asked:

ENTER THE NAME OF THE SAVE FILE:

The file name entered represents your customized version of a time-shared run-time system. The default extension is .SAV.

Your final carriage return begins a process of assembling and linking files that will produce your customized time-shared run-time system. Your answers create the file TSDPAR.MAC and the indirect file CTSGEN.COM which contains the commands for assembling and linking the time-shared run-time system. After the final carriage return is entered, CTSGEN chains to CTSGEN.COM to complete the creation of your customized run-time system. The system is identified by the file name you specified in the last question of the CTSGEN dialog.

In addition, the linker creates a link map with this same file name and a default extension of .MAP. This link map may help identify problems with a time-shared run-time system. Note in particular that the link map can tell you if an XMTSD system built as a result of choosing maximum values (or with values greater than those supported) can run. The value for the "next free address", located at the bottom of the link map, cannot exceed 100000 if the system is to run.

INTRODUCTION TO SECTION III

Section III contains the system utilities provided with CTS-300. These utilities assist you in maintaining your system, in monitoring its operation, and in providing a convenient way to present data. Each utility is discussed in its own chapter. There is no particular order of presentation. Some of the utilities are usable in all systems and some are applicable to single-user or time-shared systems only.

The utilities and their major functions are:

DDT (Chapter 7)

A run-time debugging program for DIBOL programs.

Spoolers (Chapter 8)

There are two line printer spoolers. One for single-user systems and one for time-shared systems. These programs implement the DIBOL LPQUE statement to print data.

PRINTU (Chapter 9)

A program that allows you to quickly develop and organize simple reports from a known data base structure.

ISMUTL (Chapter 10)

A program that permits you to develop an ISAM file that is tailored to your specific needs and to monitor this file and reorganize it as it grows.

SORTGEN (Chapter 11)

A program that allows you to sort or merge files using up to eight keys.

SORT (Chapter 12)

A sort/merge program released with Version 7 that has several advantages over the SORTGEN program above.

STATUS (Chapter 13)

A utility that shows the current system operation parameters such as numbers and names of jobs and terminals for a time-shared system and messages for both time-shared and single-user systems.

REDUCE (Chapter 14)

A time-shared utility used to eliminate the unused blocks of a DIBOL program file that result from linking a program for time-shared operation.

MSGUTL (Chapter 15)

A utility used to update the error message text file. MSGUTL runs only in the single-user environment.

CHAPTER 7

DIBOL DEBUGGING UTILITY (DDT)

7.1 INTRODUCTION

DDT (DIBOL Debugging Technique) is a system utility that allows you to interact with your DIBOL program while it is executing.

7.1.1 Features

The features of DDT are intended to aid the DIBOL programmer in locating problems; correcting data values; and testing programming errors directly without having to edit, compile, and link again. Specifically, you may:

- Set predetermined stopping points.
- Examine and/or alter the contents of variables.
- Single step through lines of a DIBOL program.
- Trace through sequences of XCALL nestings.

7.1.2 Chapter Organization

The remainder of this chapter is arranged in two major sections. Section 7.2, Preparing for DDT, discusses the procedures required to prepare your system and program for DDT operation. Section 7.3, DDT Commands, describes the various DDT commands and their use.

7.2 PREPARING FOR DDT

This section details the procedures required to compile, link, and run with DDT.

7.2.1 CTSGEN

You must request DDT during CTSGEN to provide DDT support at run time.

7.2.2 Compiling

The main program, as well as all subroutines which are to be debugged, must be compiled for use with DDT by specifying the DDT option in the DIBOL compiler command. This option generates a symbol table used by DDT. A typical command line might be:

```
.R DICOMP  
*PROG,PROG = PROG,SUB1,SUB2,SUB3/D
```

which generates PROG.OBJ, SUB1.OBJ, SUB2.OBJ, SUB3.OBJ, and PROG.LST. The necessary symbol tables for DDT are generated.

7.2.3 Linking

The compiled main program and all subroutines must be linked with a special DDT module in order for it to be available at run time. Notice that programs for both the single-user system and a time-shared system are linked to the same DDT module (TSDDT). The basic command for the main program and subroutines used in compiling would be:

For single-user operation:

```
.LINK PROG,SUB1,SUB2,SUB3,TSDDT,ODIBOL,DIBOL
```

which would create PROG.SAV.

For time-shared operation:

```
.LINK/EXE:PROG.TSD PROG,SUB1,SUB2,SUB3,TSDDT,ODIBOL,-  
TDIBOL/BOT:100000
```

which would create PROG.TSD.

7.2.4 DDT Operation

7.2.4.1 Running DDT — DDT is initialized whenever a program compiled and linked for DDT operation is run under a run-time system which includes DDT support. DDT outputs its version number and, on the next line, the hyphen prompt. In the following program, PROG, which has been compiled and linked for DDT, is executed for manipulation with DDT commands:

```
.R PROG or .RU dev:PROG  
DIBOL DDT VAnn-nn  
-
```

where the A in the version identifier indicates an SUD system. It would be B for a TSD system and C for an XMTSD system.

At this point any valid command discussed in Section 7.3 can be entered.

7.2.4.2 Failure to Properly Prepare for DDT — If you forget to perform one of the required steps in Sections 7.2.1 through 7.2.3, the program will exhibit the following characteristics:

- If no DDT were requested at CTSGEN time:
The program will run as though no DDT were requested.
- If no DDT were requested during compilation:
The program will respond to DDT except for those commands that examine and/or alter the contents of variables.
- If no DDT were requested during linking:
The program will run as though no DDT were requested.

7.2.4.3 Error Messages — See the *CTS-300 System Message Manual* for DDT error messages and their meanings.

7.3 DDT COMMANDS

This section discusses the commands that are valid for DDT. In the following text, when the term routine is used it refers to a specific program module; either the main program or an external DIBOL subroutine.

For future reference, the DDT commands and command formats are listed below in the order they appear in this section.

Task	Command
Start or resume execution	CTRL/Z
Single step	CTRL/G
Setting breakpoints	\$(name:)nnn
Clearing breakpoints	\$(name)
Iteration of breakpoints	> n
Examining variables	vvv=
Setting variables	vvv= nnn
Extended variable manipulation	+ + vvv= or + + vvv= nnn
Subroutine traceback	^

DDT commands (except the CTRL/Z and CTRL/G commands), must be followed by a carriage return (<CR>).

7.3.1 Program Execution Control

Program execution control has two functions: it allows you to resume execution after a breakpoint has been encountered; and it allows you to single step through individual DIBOL statements to see if they are being properly executed.

7.3.1.1 Program Execution — To start or resume execution of the DIBOL routine from a DDT breakpoint, enter the following command in response to the DDT prompt:

^CTRL/Z

There are no arguments. The current routine simply starts or resumes execution.

7.3.1.2 Single Step — It is frequently desirable to know which branch of a computed GOTO, or of a complicated IF statement the program will take. The single step command executes the next instruction in the routine and halts. To single step, enter the following command in response to the DDT prompt:

```
-CTRL/G
```

There are no arguments. The routine executes the present instruction and returns the following message:

```
AT LINE xxxx IN ROUTINE yyyy
```

```
-
```

where:

xxxx is the line number of the instruction after execution of the single step.

yyyy is the name of the routine in which line xxxx resides.

You may now enter any DDT command in response to the DDT prompt.

Example:

Assume there is a conditional GOTO statement at line 47 in subroutine SUB3, and you want to find the next instruction to be executed. First, while in subroutine SUB3, set a breakpoint (see Section 7.3.2.1) at line 47:

```
$47
```

Start execution of the routine by issuing a CTRL/Z. When line 47 is reached, the display will be:

```
DDT BREAK AT LINE 47 IN ROUTINE SUB3
```

```
-
```

Respond to the prompt with CTRL/G. The display will be:

```
AT LINE __nn IN ROUTINE SUB3
```

```
-
```

where:

nn is the location of the next instruction to be executed.

7.3.2 Breakpoint Control

A breakpoint is a user-determined stopping point within a routine. Breakpoints are used in a routine to exercise other DDT capabilities.

7.3.2.1 Setting Breakpoints — Type the following command in response to the DDT prompt to set a breakpoint:

`-${name:]nnn`

where:

name is the name of the routine in which the breakpoint is to be set. If a breakpoint is to be set in the main program, the name of the first routine specified in the link command (by convention, the root segment) should be used. Otherwise, the name of the routine should match the name given in the subroutine statement. If the name argument is omitted, the current routine is assumed.

nnn is the line number at which the routine is to halt.

- The line at which the routine is halted has not yet been executed.
- Only one breakpoint is allowed in any main program or subroutine at any given time.
- A maximum of eight breakpoints may be set at any one time.
- A breakpoint in the data section has no meaning and will never cause a break.

Example:

`-$SUB1:50`

sets a breakpoint at line 50 in subroutine SUB1.

`-$21`

sets a breakpoint at line 21 in the current routine.

7.3.2.2 Clearing Breakpoints — Previously set breakpoints may be cleared by typing the following command in response to the DDT prompt:

`-${name]`

where:

name is the name of the routine in which the breakpoint is to be deleted. If name is omitted, the breakpoint in the current routine is cleared.

- The breakpoint in a routine need not be deleted before a breakpoint is set at a new line in that routine.
- Setting a new breakpoint automatically deletes any other breakpoint in that routine.

Example:

```
-$SUB2
```

clears the breakpoint in subroutine SUB2.

```
-$56
```

sets a breakpoint at line 56 in the current routine and clears any prior breakpoint in that routine.

7.3.2.3 Iteration of Breakpoints — To test the effects resulting from iterative procedures, it is sometimes useful to set a breakpoint in a loop and pass through it several times before allowing execution to halt. This is accomplished with the following command in response to the DDT prompt:

```
->n
```

where:

n is the iteration count specifier. This is the number of times the breakpoint is to be encountered before execution is halted.

- The iteration count can be set only in the current routine.
- You must be at the breakpoint before issuing the iteration command.
- Execution is halted the nth time the breakpoint is encountered.

Example:

Assuming that a breakpoint is set in a loop at line 25 of the current routine, and the program executes until reaching this point, the response will be:

```
DDT BREAK AT LINE 25 IN ROUTINE XXX
```

```
-
```

where:

XXX is the name of the current routine.

You might respond with an iteration count and execution command:

```
->8
```

```
-CTRL/Z
```

The routine then loops through this location; stopping the eighth time it reaches line 25. The response is:

```
DDT BREAK AT LINE 25 IN ROUTINE XXX
```

```
-
```

7.3.3 Variable Manipulation

Variable manipulation allows you to change or examine variables in a routine to determine whether or not they are being correctly handled.

7.3.3.1 Setting Variables — Variables may be set (loaded) with any desired value by using the following command in response to the DDT prompt:

```
-vvv = nnn
```

where:

vvv is the variable name.

nnn is the value you want to assign to the variable.

- If the length of nnn is too long to store in vvv, the data is left justified in the field and the excess right-hand characters are truncated. This is true for both alpha and decimal fields.
- Do not use single quotes when specifying alpha data.
- A field, alpha or decimal, can be cleared by entering a space for an assigned value.
- Decimal literal subscripts, single or double, can also be used to set variables.

Examples:

```
-VAR1 = ABCD
```

Assigns the value of ABCD to VAR1.

```
-VAR1 = 'ABCD'
```

Assigns the value of 'ABC to VAR1.

7.3.3.2 Examining Variables — Variables may be examined to verify their contents with the following command in response to the DDT prompt:

```
-vvv =
```

where:

vvv is the variable name. Decimal literal subscripts, either single or double, may be used with the variable name to access a part of a field or data in an unlabeled field.

Example:

Assume you have stopped at a breakpoint; then:

```
-VAR1 =
```

results in a display of the present contents of this variable.

7.3.3.3 Extended Variable Manipulation — It is possible under the specific circumstances explained here to examine, or to set, a variable used outside the current routine. This may be done only when the variable is defined in the routine which called the current routine or is defined in one of the routines in the chain of calls which led to the current routine. For example:

- + + VAR2 =

will return the current value of VAR2 located in the chain of routines which called the current routine. The two plus signs indicate that the variable was defined in a routine located two calls back (two levels of nesting) in the chain which led to the current routine. Also:

- + + VAR2 = EFGH

will set VAR2 to the value EFGH.

7.3.4 Subroutine Traceback

The subroutine traceback feature allows you to determine whether or not the calling sequences (XCALL statements) are executing in the expected manner. The output is a list of the routines and the line numbers in those routines of all the related preceding XCALL statements back to the main program. To obtain this list, enter the following command (a caret, up arrow, or circumflex) in response to the DDT prompt:

- ^

There are no arguments. The circumflex (^) causes the list to be generated.

Example:

Assume you have halted in a subroutine at a DDT breakpoint, or you have single stepped to the current position, and you need to know how you arrived at this point from the main program. The command and traceback list might look like the following:

```
- ^
AT LINE 37 IN ROUTINE SUB3      (current location)
AT LINE 192 IN ROUTINE SUB2    (SUB3 called from SUB2, line 192)
AT LINE 21 IN ROUTINE MAIN     (SUB2 called from MAIN, line 21)
```

You are still in routine SUB3 and may enter any DDT command.

CHAPTER 8

LINE PRINTER SPOOLERS

8.1 INTRODUCTION

The two line printer spoolers, LPTSP1.REL and LPTSPL.TSD, are utility programs used to print data and program source files. LPTSP1.REL is used with SUD programs, and LPTSPL.TSD is used with TSD and XMTSD programs. Besides interfacing to a DIBOL program via the LPQUE statement, LPTSPL.TSD can also interface directly with the operator via a program called QUE.TSD.

8.1.1 Common Characteristics

Both spoolers operate in response to the DIBOL LPQUE statement. The function of the spooler, regardless of run-time system, is to queue files to be printed, and to issue print commands to the printer.

If a line printer spooler is to be used, it must be the only program to output to the printer(s). Programs requiring printer output should do so via the spooler.

8.1.2 Chapter Organization

The remainder of this chapter is comprised of two sections: Section 8.2, Single-User Systems, and Section 8.3, Time-Shared Systems. Each section is organized the same way: first, the characteristics of the particular spooler are covered; then the details associated with using the spooler are presented.

8.1.3 Error Messages

Error messages associated with LPTSP1.REL, LPTSPL.TSD, and QUE.TSD are documented in the *CTS-300 System Message Manual*. See Section 8.2.5 for your required response to errors that occur while running LPTSP1.REL.

8.2 SINGLE-USER SYSTEMS (LPTSP1.REL)

8.2.1 LPTSP1.REL Characteristics

Features

The SUD spooler has the following features:

- The SUD spooler is a queue manager program written in assembly language.

- Shared operation of the line printer (between the spooler and a DIBOL program) is not supported. See Section 8.2.2.1 for further information.
- The SUD spooler operates as a foreground job. Other SUD programs run concurrently in the background. See Section 8.2.2.2 for further information.
- Output is to one line printer only.
- LPTSP1.REL can queue up to ten print jobs.

Requirements

The SUD spooler requires the following system supports:

- LPTSP1.REL requires 4 K bytes of memory.
- A handler, LP.SYS, must be resident (via the RT-11 LOAD command) in memory.
- LPTSP1.REL must be on the system device.
- LPTSP1.REL requires either the FB or the XM monitor.

8.2.2 Using LPTSP1.REL

8.2.2.1 Shared Line Printer Operation — The line printer cannot be shared between the line printer spooler and a DIBOL program or a CTS-300 system program. An attempt to use the line printer while LPTSP1 is running will result in a device-in-use error message.

8.2.2.2 Shared Terminal Operation — Both the line printer spooler and a DIBOL program running concurrently under the FB monitor can make use of the same terminal. As stated above, the spooler runs in the foreground, and the DIBOL program runs in the background.

I/O operations at the terminal are independent functions for the two programs. The line printer spooler can display a message at any time by assuming control of the terminal. Data that is input at the terminal while the spooler is using the terminal is stored in a terminal buffer. When the spooler releases control, the stored input data is displayed. A symbol is displayed with each message to the terminal to identify the message source. When the foreground job outputs to the terminal, the message is preceded by F> . When a background job outputs to the terminal it is preceded by B> . Likewise, an indicator must be provided by the operator to identify the program being addressed; CTRL/F to direct input to the foreground, CTRL/B to direct input to the background. The use of these symbols is shown where necessary in the following text.

If both the spooler and the DIBOL program require the terminal simultaneously, the spooler has priority because the foreground job always has priority. Under these conditions, data output from the spooler continues until it is done. Control then passes to the DIBOL program until the spooler again has a need for the terminal.

8.2.3 Starting LPTSP1.REL

If you want to print a file that is not on the system device, you must load the handler for that device to make it available for the spooler.

```
.LOAD dev
```

where:

dev is the name of the device where the file to be printed resides.

The spooler is started by entering the following series of commands (The handler is loaded for the exclusive use of the foreground program — the spooler. This avoids mixing output from the spooler with output from a DIBOL program):

```
.FRUN LPTSP1  
.LOAD LP=F  
.RESUME LPTSP1
```

The response is:

```
F>  
SINGLE-USER DIBOL LINE PRINTER SPOOLER VAnn-nn  
B>
```

where:

nn-nn is the version number for the spooler program.

When the monitor's prompt (.) appears, any runnable DIBOL program can be executed. An LPQUE statement within that program will then cause the spooler to queue and print the specified file.

8.2.4 Run-Time Dialog

If the FORM argument were included in the LPQUE statement, the spooler will output an appropriate message to the terminal. The message is preceded by F> to indicate that it has been issued by the foreground job (LPTSP1). The message and appropriate response is shown below:

Message

```
F> Load forms xxxxxx on line printer LP:
```

Response

The operator must load the specified form (xxxxxx), that is, invoices, payroll checks, etc., into the line printer. Operation is continued by typing CTRL/F <CR> . CTRL/B is then used to preface input to the background.

8.2.5 Response to Error Messages

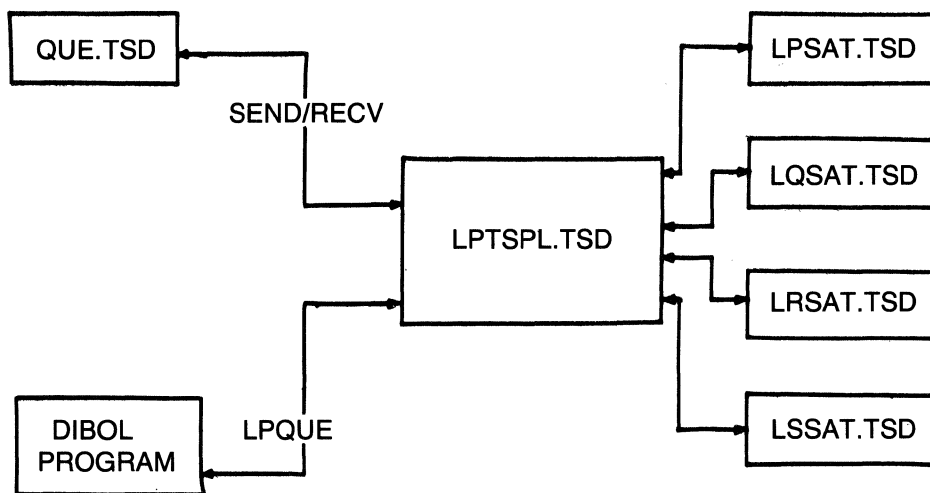
All errors for LPTSP1.REL are fatal errors. If an error occurs, any queued file(s) is lost, and you must then take the following steps:

1. Enter at the terminal:
`.UNLOAD F`
2. Correct the condition that caused the error.
3. Restart the spooler, as described in Section 8.2.3.
4. Request a print again.

8.3 TIME-SHARED SYSTEMS (LPTSPL.TSD and QUE.TSD)

Spooling for printer output in a time-shared system is handled basically by two components. These are the QUE.TSD program providing the terminal interface and the spooler program LPTSPL.TSD which controls the spooler satellites.

The relationship of LPTSPL.TSD to QUE.TSD and to DIBOL programs is shown below:



8.3.1 LPTSPL.TSD Characteristics

The Queue File

LPTSPL maintains a list of files to be printed in file LPQFIL.LQP on DK:. There are a total of 100 queue entries possible in LPQFIL.LPQ, with no fixed number of entries assigned to a given printer. If the queue is full when a print request is received from QUE.TSD, the spooler notifies QUE.TSD that the queue is full, and that the file was not accepted for printing. If the request was via an LPQUE statement, the notification would be sent to a log file (see the QUE LIST command, Section 8.3.9.6).

File Recovery

If spooler execution is terminated before the completion of all print requests, the remaining requests are stored in file LPQFIL.LQP. When the spooler is restarted, it checks this file and makes available the status of the print requests to QUE.TSD. It is good practice under such conditions to run QUE to determine partially completed or remaining print jobs so they can then be printed.

Suspension of Spooling

When the spooler is started, it opens an I/O channel to the line printer(s). If the printer(s) is being used by another user, spooler operation is suspended for the busy printer until the other user releases the printer.

Default Printers

A printer (or more, up to a total of four) can be designated as a default printer by the user. This is done when LPTSPL.TSD is first run (see Section 8.3.3). If a print job is received that is not assigned to a specific printer, it is queued by LPTSPL.TSD to the default printer. If more than one printer were designated as a default printer, the job is queued to the first free default printer.

General Characteristics

- LPTSPL.TSD is a DIBOL program that operates under the control of a time-shared run-time system.
- LPTSPL.TSD is the queue manager and has associated with it four satellite programs: LPSAT.TSD, LQSAT.TSD, LRSAT.TSD, and LSSAT.TSD. Each satellite program is active only as long as there are files to be printed on its associated printer.
- Each satellite automatically opens its line printer.
- A maximum of four line printers is supported.
- LPTSPL.TSD supports assignment of default line printers. See Section 8.3.3 for more information.
- If spooler execution is properly terminated (see the QUE STOP command, Section 8.3.9.11) before completion of all print requests, the remaining requests are stored for later recovery. The current print job is completed before the spooler is terminated.
- LPTSPL.TSD runs detached.

8.3.2 LPTSPL.TSD Requirements

LPTSPL.TSD requires the following system supports:

- Before execution of LPTSPL.TSD in the XMTSD environment, the appropriate handler(s) must be loaded. If all four handlers are required, the procedure is:

```
.LOAD LP
```

```
.LOAD LQ
```

```
.LOAD LR
```

```
.LOAD LS
```

or you may issue a single line command:

```
.LOAD LP,LQ,LR,LS
```

Under the XMTSD monitor, the correct handler is loaded with the above commands. This is done despite the fact that the XM handler name has a third character (X). After the handlers are loaded, proceed as described in Section 8.3.3 to run the spooler.

Under TSD the handlers are automatically loaded.

If you want to print a file that is not on the system device, you must also load the handler for that device to make it available for the spooler. Use the following command:

```
.LOAD dev
```

- The LPTSPL.TSD queue manager requires 6 K bytes of memory; each satellite requires an additional 2 K bytes.
- The number of line printers must be specified during SYSGEN.
- Forced job startup must be requested during CTSGEN in order to utilize LPTSPL.TSD.

8.3.3 Running LPTSPL.TSD

The line printer spooler, LPTSPL.TSD, may be started by any one of four methods:

1. Direct Startup
2. Forced Job Startup
3. Chain Mode Startup
4. Implicit Job Startup

Comments on Methods 2, 3, and 4 (below):

When LPTSPL.TSD is started as a detached program, the version number message is not displayed, and default printer selection is not asked. The default printer selection information can be supplied via a DIBOL SEND statement from the DIBOL program which is initiating startup of LPTSPL.TSD. This is illustrated with each of the three startup methods in Sections 8.3.3.2, 8.3.3.3, and 8.3.3.4. In these, the question mark in the SEND statement message SEND('?YNNY',...) indicates to the spooler that the information defines default line printers. Each character after the question mark is a response to a default printer selection question (yes, no, no, yes).

If the default printer selection information is not supplied in a SEND statement, LPTSPL.TSD automatically designates LP: as the default printer.

8.3.3.1 Direct Startup (Method 1): — LPTSPL.TSD can be started under TSD or XMTSD like any other program simply by specifying its name in response to the asterisk prompt:

```
*LPTSPL
```

When LPTSPL.TSD is started, the following message is displayed:

```
Time-Shared DIBOL Line Printer Spooler VBnn-nn
```

where:

nn-nn is the version number.

The spooler then asks for default printer identification:

```
— Default Line Printers —  
Line Printer LP: <Y/N> ? Y
```

If line printer LP: is to be a default printer, enter Y (or <CR>). If this printer is not to be a default printer, enter N. The remaining printers (LQ:, LR:, and LS:) are then individually presented for default selection:

```
Line Printer LQ: <Y/N> ? N  
Line Printer LR: <Y/N> ? N  
Line Printer LS: <Y/N> ? N
```

The default answer for each of these remaining printers is N. You must respond for each printer.

Note that the printer assignment here in the order LP:, LQ:, LR:, LS: corresponds to printers 1, 2, 3, and 4 as specified in the LPQUE statement (see Section 8.3.4).

After default printer selection questions are answered, the spooler detaches and returns you to the run-time system asterisk prompt. At this point you will probably run QUE.TSD (see Section 8.3.5).

8.3.3.2 Forced Job Startup (Method 2): — LPTSPL.TSD can be started from within a program using forced job startup:

```
XCALL RUNJB ('LPTSPL.TSD',term#)
```

or, if a -1 is specified as the terminal number (indicating a detached job), the spooler can be supplied with the default printer selection information as follows:

```
SEND ('?YNNY','LPTSPL.TSD')  
XCALL RUNJB ('LPTSPL.TSD',-1)
```

8.3.3.3 Chain Mode Startup (Method 3): — LPTSPL.TSD can be started from within a program using chain mode startup:

```
STOP 'LPTSPL.TSD'
```

If the spooler is started up with an attached terminal, the default printer selection questions are asked in the same manner as when the spooler is started via keyboard command. After the questions are answered, LPTSPL.TSD detaches itself. If the chaining job is detached, the spooler can be supplied with the default printer selection information as follows:

```
SEND ('?YNNY','LPTSPL.TSD')  
  
STOP 'LPTSPL.TSD'
```

8.3.3.4 Implicit Job Startup (Method 4): — LPTSPL.TSD can be started from within a program using implicit job startup:

```
SEND ('?YNNY','LPTSPL',-2)
```

8.3.4 Response to the LPQUE Statement

Ordinarily, a specific line printer is assigned by line printer number to a print job from within a DIBOL program. For example, the following LPQUE statement assigns the second line printer as the printer to print the file named XXX.YYY:

```
LPQUE('DL1:XXX.YYY',LPNUM:2)
```

However, this is not a requirement. If no printer is specified, the default printer is the one specified when LPTSPL.TSD was first run.

If the FORM argument were included in the LPQUE statement, printing will be suspended until the correct form is loaded and the spooler notified. Notification of LPTSPL.TSD is done through use of the QUE.TSD program (see the next section).

8.3.5 QUE.TSD

The QUE.TSD system program accepts requests to be executed by the spooling program, LPTSPL.TSD. QUE also accepts commands to perform such functions as listing pending requests (LIST), terminating requests (KILL, FLUSH), modifying pending requests (MODIFY), and interrupting requests being spooled (INTERRUPT). The QUE program checks the syntax and validity of each request and passes it to the spooler for further processing. If any part of the request is invalid, QUE rejects the entire request, prints an appropriate error message, and allows you to try again.

QUE has available four HELP frames that present you with concise information on commands and options used with QUE.

8.3.6 Running QUE.TSD

There are two ways in which QUE.TSD can be used to enter print requests and to issue QUE commands.

QUE.TSD can be executed in response to the TSD or XMTSD asterisk prompt:

```
*QUE <CR>
```

the response is the QUE prompt:

```
QUE >
```

to which you respond with the command and/or file specification. Once processing of the command is complete, you remain in the QUE program and further QUE commands can be issued. This method of using QUE is, for the remainder of this chapter, referred to as operating at the QUE program level.

The other method of using QUE.TSD is to include commands, options, and file specifications in a single command string along with the request to execute QUE.TSD.

The single-line command string form is:

```
*QUE/command[/option(s)] <CR>
```

Once the processing of this string is complete, QUE returns you to the time-sharing system prompt. This method of using QUE will be referred to as operating at the TSD/XMTSD level.

LPTSPL.TSD must be running before QUE.TSD is executed, otherwise a message will appear stating that the spooler is not running.

8.3.7 QUE.TSD Options

There are eleven options that can be used with QUE. Some of these options are valid when you are specifying files to be queued and some are valid when referring to files already in the queue. This is an overview of the options, and they are presented here separate from any specific use. Later they will be shown in connection with file queuing and with specific QUE commands.

Option	Description
ASSIGN	Specifies that the job is to be transferred to another spooler queue.
COPIES	Specifies the number of copies of the file to be printed.
DELETE	Causes deletion of the file after it is printed.
NODELETE	Cancels the DELETE option.
DEVICE	Specifies the output device (printer).
FORMS	Identifies a form on which the job is to be printed.
HOLD	Specifies that the job is to be held in place in the queue.
NOHOLD	Specifies that a job previously held in queue is to be released.
KILL	Specifies that the printing of a file be terminated and that the queue entry for that file be removed.
PAGES	Specifies the pages to be printed. Starting and ending pages can be identified.
PRIORITY	Assigns print priority.

8.3.8 Queuing Files for Print

A request to queue a file for print is probably easiest to do using QUE at the TSD/XMTSD level; this is the method that will be shown.

Once a file (or files) has been queued, you can, by using the commands and appropriate options, change the original specifications of the files in queue. You can obtain a list of the files in queue for any of the printers and make changes to the print parameters to any file still in queue. Other commands allow you to eliminate a file from queue, eliminate all the files in a queue, stop the spooler and satellites in a controlled manner, look at the error log, and terminate QUE.TSD operation. All these operations are discussed in Section 8.3.9, QUE Commands.

Specifying QUE with a filespec (and valid options) creates a request for a file to be spooled. The following information is summarized in the first help frame (see Section 8.3.9.4).

The form is:

```
*QUE[/option(s)] filespec
```

where:

filespec is the file specification of the file to be spooled.

/options(s) is one or more options from the following list:

Option	Description
<code>/C[opies] = nnn</code>	Specifies the number of copies of the file to be printed where nnn is the number of copies. The default is one copy.
<code>/DEL[ete]</code>	Causes deletion of the file after it is printed.
<code>/DEV[ice] = xx</code>	Specifies the output device (printer). The value of xx can be LP, LQ, LR, or LS. If omitted, the default is DF.
<code>/FO[rms] = zzzzzz</code>	Identifies a form on which the job is to be printed. The string zzzzzz is the name of the form and can be up to six characters long. Specifying a form will cause the file to be held until operator confirmation is received that the form is loaded. See Section 8.3.10 and the TALK command in Section 8.3.9.12.
<code>/HO[ld]</code>	Specifies that the job is to be held in place in the queue and is not to be sent to a spooling program until the MODIFY command with the /NOHOLD option is exercised.
<code>/PA[ges] = nnn[,mmm]</code>	Specifies the pages to be printed. Printing starts at page nnn and proceeds to page mmm, inclusively. If mmm is not specified, printing proceeds to the end of the file. Page boundaries are determined by form-feed characters.
<code>/PR[iority] = n</code>	Assigns print priority. The value of n can be 1 through 9, inclusive, with 9 being the highest priority. The default is a priority of 5. A higher priority assignment will ensure that a job will be printed before a job of lower priority.

Examples:

```
*QUE/C = 2/DEV = LR/PA = 5,10 DL0:REPORT.TXT
```

Prints on line printer LR two copies of pages 5 through 10 of REPORT.TXT which is on device DL0.

```
*QUE/DEL/FO = CHECK/PR = 9 PAYCHK.DDF
```

Prints on a default line printer one copy of PAYCHK.DDF which is on the system default device. The form CHECK must be loaded and confirmed using the TALK command. This job will print before others which have a priority less than 9. The file will be deleted after it is printed.

8.3.9 QUE Commands

There are four categories of QUE commands.

Command	Operation
1. /EXIT /STOP	directed to QUE.TSD
2. /ERROR /FLUSH /LIST /PURGE /MODIFY	is directed to the queue file, LPQFIL.LPQ
3. /INTERRUPT /RESERVE /RELEASE /TALK	is directed to the satellites
4. /HELP	displays the help frames

Commands are probably easiest to enter when at the QUE program level (when the QUE prompt is displayed). This is the method that will be used in the following discussion of QUE commands. The commands are listed in alphabetical order.

Option characteristics:

- Options can be specified anywhere in the command line. For consistency they are shown in this chapter following the command.
- If the command allows multiple file specifications, specified options apply to all files.

8.3.9.1 ERROR — The ERROR command specifies a listing of the error log file. It lists the last five errors and prompts for further display. There can be as many as 50 errors. Errors in addition to the five shown can be seen by pressing <CR> .

The form is:

```
QUE> /ER[ror]
```

There are no arguments or options.

Example:

```
QUE> /ERROR
```

QUE ERROR LISTING

```
-----  
                08-May-81    00:06:33  
E6—Device handler not available — (XXX)  
-----  
                08-May-81    00:05:30  
E7—File of length zero — (DL0:REPORT.OLD)  
-----  
                08-May-81    00:05:03  
E11—File not found — (DL1:LOST.FIL)  
-----  
                08-May-81    00:04:45  
E9—Input from write only device — (LP)  
-----  
                08-May-81    00:02:33  
E5—Line printer handler not available — (LQX)  
-----
```

Continue to the next frame <Y/N> ? (Y)

This example shows typical errors that can be logged. Note that the most recent error is listed first.

8.3.9.2 EXIT — The EXIT command terminates QUE.TSD and returns control to the time-shared run time system.

The form is:

```
QUE> /EX[it]
```

There are no arguments or options.

8.3.9.3 FLUSH — The FLUSH command removes all jobs in a queue. You are prompted for verification of the flush.

The form is:

```
QUE> /FL[ush][/DEV[ice]= xx]
```

where:

/DEV[ice]= xx

Specifies the output device (printer) associated with the queue to be flushed. The value of xx can be LP, LQ, LR, LS, or DF.

If /DEVICE is omitted, the entire queue is flushed - including DF:.

8.3.9.4 HELP — The HELP command causes the first of four HELP frames to be displayed. These frames provide a summary of the commands and options available in QUE.TSD.

Each frame (except the last) asks you if you want to continue to the next. A Y response (the default) selects the succeeding help frame. A N response gives you the opportunity to enter a command.

The form is:

QUE> /HE[lp]

There are no arguments or options.

Help Frame 1

To PRINT a file enter filename(s) with or without options
 — options will hold throughout all filespecs

 FORMAT — QUE[/options] filespec(s)

options :

- /C[opies] = nnn
- /DEL[ete]
- /DEV[ice]-xx
- /PR = n
- /FO[rms] = zzzzzz
- /HO[ld]
- /PA[ges] = nnn[,mmm]

where :

- xx or yy.....LP, LQ, LR, LS or DF
- zzzzz.....six character form name
- nnn or mmm.....three digit number
- n.....one digit number

Continue to the next frame <Y/N> ? (Y)

Help Frame 2

Summary of /COMMANDS and valid /OPTIONS.

FORMAT — QUE/command[/options][filespec(s)]

 command : valid options

/ER[ror]	/DEV[ice] = xx
/EX[it]	
/FL[ush]	/DEV[ice] = xx
/H[elp]	
/I[nterrupt]	/DEV[ice] = xx, /K[ill], /PA[ges] = nnn[,mmm]
/L[ist]	/DEV[ice] = xx
/M[odify]	/A[ssign] xx yy, /C[opies] = nnn, /DEL[ete], /NOD[ete], /DEV[ice] = xx, /FO[rms] = zzzzzz, /HO[ld], /NOH[old], PA[ges] = nnn[,mmm], /PR = n
/PU[rge]	/DEV[ice] = xx
/REL[ease]	
/RES[erve]	
/S[top]	
/T[alk]	/DEV[ice] = xx

Continue to next frame Y/N ? (Y)

Help Frame 3

COMMANDS

/ER[ror].....	List the error log contents (**)
/EX[it].....	Terminate QUE.TSD
/FL[ush].....	Remove all jobs in a queue (**)
/M[odify].....	Modify options on a file in the queue (*)
/HE[lp].....	Display HELP frame at the terminal
/I[nterrupt].....	Abort or restart a spooling process (**)
/L[ist].....	List the contents in queue (***)
/PU[rge].....	Remove a file from the queue (*)
/REL[ease].....	Cancel the RESERVE command (***)
/RES[erve].....	Free up an LP (***)
/S[top].....	Stop LPTSPL after LP's finish present job
/T[alk].....	Communicate between user and printer (**)

note — One command per line preceded by a “/”
* No device specified implies the default DF
** No device specified implies all queues
*** No device specified user will be prompted

Continue to the next frame <Y/N> ? (Y)

Help Frame 4

OPTIONS

/A[ssign] xx yy.....	File(s) transferred from xx to yy
/C[opies] = nnn.....	Number of copies to be printed
/DEL[ete].....	File(s) to be deleted after printing
/NOD[elete].....	Cancel the /DELETE option
/DEV[ice] = xx.....	Specifies the output device
/FO[rms] = zzzzzz.....	Form job is to be printed on
/HO[ld].....	Job to be held in queue
/NOH[old].....	Cancel the /HOLD option
/K[ill].....	Abort the job being spooled
/PA[ges] = nnn[,mmm].....	Print pages nnn through mmm
/PR[iority] = n.....	Priority value of 1 through 9

where:
xx or yy.....LP, LQ, LR, LS or DF
zzzzzz.....six character form name
nnn or mmm.....three digit number
n.....one digit number

QUE>

8.3.9.5 INTERRUPT — The INTERRUPT command allows you to interrupt a spooling process to either abort or restart printing. It can take up to 20 lines before printing stops.

The form is:

```
QUE> /I[nterrupt]/DEV[ice] = xx[/option(s)]
```

where:

```
/DEV[ice] = xx
```

Specifies the output device (printer). The value of xx can be LP, LQ, LR, or LS.

- Device DF is not allowed.
- If no value for xx is supplied, you are prompted for the device.

```
/option(s)
```

is one of the options listed below.

Option	Description
--------	-------------

/K[ill]	Specifies that the currently printing file be terminated and that the queue entry for that file be removed.
---------	---

```
/PA[ges] = nnn[,mmm]
```

Specifies a new selection of pages to be printed. This new selection applies only to the job currently printing. Other copies of this file or other files will use the originally specified page parameters. Printing starts at page nnn and proceeds to page mmm, inclusively. If mmm is not specified, printing proceeds to the end of the file. Page boundaries are determined by form-feed characters.

8.3.9.6 LIST — The LIST command lists printer queue contents.

The form is:

```
QUE> /L[ist]/DEV[ice] = xx
```

where:

```
/DEV[ICE] = xx
```

Specifies the output device (printer) associated with the queue to be listed. The value of xx can be LP, LQ, LR, LS, or DF.

- If omitted, the queue for each printer (including DF) having files queued is displayed one at a time in the order LP, LQ, LR, LS, DF.

Examples:

Queue listing (for device LP):

```
QUE> /LIST/DEV = LP
```

Q U E L I S T I N G

LINE PRINTER LP:							
file name	delet	forms	copies	start	end	prior	status
DL1:PAYROL.DDF:	N	:CHECKS:	1	:	:	: 7	:Awaiting /TALK command
DK:LIST.DDF	: Y	:	: 1	:	:	: 5	:Being held

QUE>

This example shows the first file queued for LP: is a payroll file for printing checks. It is waiting for the correct form (CHECKS) to be loaded. The TALK command is used to confirm loading. The second file queued is to be deleted after it is printed, but will not be printed until the /NOHOLD option is specified for this file using the MODIFY command.

Queue listing (for device LR):

QUE> /LIST/DEV = LR

Q U E L I S T I N G

LINE PRINTER LR:							
file name	delet	forms	copies	start	end	prior	status
DL0:REPORT.TXT:	N	:	2	: 20	: 35	: 5	:

QUE>

The request queued for LR: is for two copies of 16 pages of a report.

Queue listing (for device DF):

QUE> /LIST/DEV = DF

Q U E L I S T I N G

LINE PRINTER LR:							
file name	delet	forms	copies	start	end	prior	status
DL0:REPT01.TXT:	Y	:	1	: 1	: 50	: 5	:

QUE>

The file queued for default printing is for one copy of file REPT01.

8.3.9.7 MODIFY —The MODIFY command allows you to modify the options of a previously specified file already in the queue but not yet being spooled. The file could have been specified by either the LPQUE statement or a QUE request.

The form is:

```
QUE> /M[odify][/option(s)][ filespecs]
```

where:

filespecs is the file specification of the file(s) to be modified. If more than one file is specified, they must be separated by commas.

- If no file specification is given, you are prompted for one.

/option(s)

is one or more of the options listed below.

Option	Description
<i>/A[ssign] xx yy</i>	Specifies that the job is to be transferred to another spooler queue. The present queue is identified by <i>xx</i> , the new que by <i>yy</i> . The value of <i>xx</i> or <i>yy</i> can be: LP, LQ, LR, LS, or DF.
<i>/C[opies] = nnn</i>	Specifies the new number of copies of the file to be printed where <i>nnn</i> is the number of copies. The default is one copy if this option is not used.
<i>/DEL[ete]</i>	Causes deletion of the file after it is printed.
<i>/NOD[etele]</i>	Cancels the /DELETE option specified earlier.
<i>/DEV[ice] = xx</i>	Specifies the output device (printer). The value of <i>xx</i> can be LP, LQ, LR, LS, or DF. If omitted, the default is DF.
<i>/FO[rms] = zzzzzz</i>	Identifies a form on which the job is to be printed. The string <i>zzzzzz</i> is the name of the form and can be up to six characters long. Specifying a form will cause the file to be held until operator confirmation is received that the form is loaded. See Section 8.3.10 and the TALK command in Section 8.3.9.12.
<i>/HO[ld]</i>	Specifies that the job is to be held in place in the queue and is not to be sent to a spooling program until the MODIFY command with the /NOHOLD option is exercised.
<i>/NOH[old]</i>	Specifies that a job previously held in queue with the /HOLD option is now to be released to proceed normally through the queue to a spooling program.

`/PA[ges]=nnn[,mmm]`

Specifies a change in the pages to be printed. Printing starts at page nnn and proceeds to page mmm, inclusively. If mmm is not specified, printing proceeds to the end of the file. Page boundaries are determined by form-feed characters.

`/PR[iority]=n`

Assigns a new print priority. The value of n can be 1 through 9, inclusive, with 9 being the highest priority.

8.3.9.8 PURGE — The PURGE command causes removal of a file from the queue. If the file is being printed, it can take up to 20 lines of print before printing stops.

The form is:

```
QUE> /PU[rge][/DEV[ice]=xx][ filespecs]
```

where:

`/DEV[ice]=xx`

Specifies the output device (printer). The value of xx can be LP, LQ, LR, LS, or DF. If omitted, the default of DF is assumed.

`filespecs` is the file specification of the file(s) to be removed from the file.

- If no file specification is given, you are prompted for one.

8.3.9.9 RELEASE — The RELEASE command cancels a previously issued RESERVE command (see below).

The form is:

```
QUE> /REL[ease]
```

There are no arguments or options.

8.3.9.10 RESERVE — The RESERVE command removes from use by the spooler a specified line printer so that a DIBOL program can temporarily use that line printer. A response is returned to let you know if the reserve was successful or not. A reserved line printer is reported in via the LIST command and is shown in the General Status Report. RESERVE is useful when FORMS is used or when you want to control loading of a form yourself.

The form is:

```
QUE> /RES[erve]
```

There are no arguments or options.

8.3.9.11 STOP — STOP is used to bring LPTSPL.TSD and the satellites to a controlled stop. The line printers are allowed to complete their printing tasks.

The form is:

```
QUE> /S[top]
```

There are no arguments or options.

8.3.9.12 TALK — The TALK command allows communication with the printing job to confirm forms loaded, reprinting of a file, or line printer hung queries. TALK is issued in response to the message in the status column of a QUE LIST or to a message in the Status Report (see Section 8.3.10).

The form is:

```
QUE> /T[alk][DEV[ice]= xx]
```

where:

```
/DEV[ice]= xx
```

Specifies the output device (printer). The value of xx can be LP, LQ, LR, or LS. Device DF: is not allowed.

The response to the TALK command can be one of the following:

For a file awaiting loading of a form:

```
Load forms xxxxxx in lineprinter xx Press <RETURN> when ready to continue
```

For file awaiting possible reprint:

```
Reprint file xxxxxxxxxxxxxxxx on lineprinter xx <Y/N> ? (Y)
```

For a hung line printer:

```
Lineprinter is hung — xx
```

8.3.10 Other Features

Status Report

If printing has been interrupted, a form has been specified, or the printer becomes unavailable, a "General Status Report" will be displayed when QUE is again run. The following shows the status report form and representative information it can display:

GENERAL STATUS REPORT

```
-----  
LINE PRINTER — LP:      Awaiting reprint confirmation  
                          This line printer is RESERVE'd  
-----  
LINE PRINTER — LQ:      Awaiting loading of FORMS  
                          This line printer is RESERVE'd  
-----  
LINE PRINTER — LR:      Awaiting operator intervention  
                          This line printer is RESERVE'd  
-----  
LINE PRINTER — LS:      Handler not available  
                          This line printer is RESERVE'd  
-----  
?SPOOLR-E15———F-Cannot create queue file LPQFIL.LPQ  
?SPOOLR-E20———E-Unable to access LPQLOG.LPQ  
-----
```

LPTSPL is coming down

Press <RETURN> when ready to continue

Trouble Shooting

If you should queue a file for print and printing does not occur, do the following: Run QUE and select the /LIST and then the /ERROR command to make sure that the printer is available, that the file is not awaiting intervention via the TALK command, that the file is not on hold, that the file existed as specified, and that the file storage device handler or printer handler is present.

CHAPTER 9

PRINTU

9.1 INTRODUCTION

PRINTU is a utility program for the creation of simple report programs using data from either a sorted sequential file or from an ISAM file. PRINTU utilizes a user-written control file that describes the parameters of the desired report. Given the control file, PRINTU generates a DIBOL program that is compiled and linked like any other DIBOL program. The resulting program, when run, produces the report.

9.1.1 Features

PRINTU has a number of features that makes it particularly useful:

- PRINTU has the ability to process data files without physically reordering them by using a sorted tag file generated by the SORT or SORTGEN utility.
- Once a report program is created, it can be stored for later use.
- Since PRINTU generates a DIBOL program, extra features or capabilities can be added by simply modifying the program to achieve your needs. Such a need could be special output print lines, more header lines, output based on logical test results, operation with packed records as input, or any other feature you want to develop.

9.1.2 Limitations

There are limitations that require foresight when you are using PRINTU. These concern the way you handle file control records (explained in the next subsection) and the DIBOL requirement for an end-of-file (EOF) marker (a CTRL/Z) in all data files. Potential problems in both of these areas are easily avoided, however.

Some application system files are designed to contain a control record as the first record. This will probably produce meaningless output on the first report page printed. There are two ways to prevent this from happening.

The first way is to do a tag sort on the file and ensure that the control record sorts out as the first record in the tag sort file. You can then use an editor to delete this record. PRINTU will then begin with the second record (valid data) in the file.

The second way to prevent a control record from interfering with a print report is to place the control record at the end of the file. To prevent PRINTU from accessing this record you can precede this record with another record containing an EOF marker as the first character. These two records can be inserted using direct access. The EOF character is obtained by using the decimal equivalent of CTRL/Z (026) and an XCALL to ASCII to insert the CTRL/Z character. DKED can also be used to insert the CTRL/Z character. The control record is then not accessible by PRINTU or via sequential access.

9.1.3 Chapter Organization

The remainder of this chapter is comprised of two sections. Section 9.2, The Control File, is a detailed discussion of the PRINTU control file and how the individual control statements relate to each other and to the desired printed report. Section 9.3, Using PRINTU, illustrates the manner in which the control file is used with the PRINTU program to produce a DIBOL file, and how this DIBOL file is compiled and linked.

9.1.4 Error Messages

Error messages associated with PRINTU are documented in the *CTS-300 System Message Manual*.

9.2 THE CONTROL FILE

PRINTU depends on information supplied by the user to describe the input file(s), output file, and other parameters of the report. This information is given to PRINTU via a control file containing control statements. The control file is created using an editor. The control file is the heart of PRINTU. It is here that you provide the parameters that determine the appearance of the report.

The control file has its own terminology and, at times, this can be confusing. For this reason, the following essential terms are introduced. Many of them are further defined in context.

accumulation field

A user-identified field whose value from record to record is accumulated to produce a total.

break field

When the data in a break field changes, a print occurs. The break field is covered in detail in Section 9.2.7.

detail print

The normal print mode (as opposed to a summary print-see below). Individual values are printed as well as the totals.

input data file

The file containing the data which is to be used as the source for your report.

report file

The output of the report program when that program is run.

report program

The program generated by running PRINTU and then compiling and linking the resulting DIBOL source file.

summary print

A printout that includes only the totals from each accumulation field whenever a new value occurs in a break field.

tag file

The file produced by the SORT or SORTGEN utility when the TAG option is selected. Used by PRINTU to access the input data file.

There are nine possible control statements in the control file, of which only four are required. The other five control statements supply optional capabilities that you may or may not need. The control file structure, format, content, individual control statement descriptions, and other characteristics are explained below and in subsequent sections.

Before we discuss the control file, briefly consider what your goals for the report are. You begin with a file whose record structure is known to you, and your major goal is to produce a printed report that shows the data in this file in some desired order in relationship to the data in each record field or fields. Additionally, there may be some data in each record that you want to ignore, and there are probably one or more fields in related records which contain numeric data that you would like to have totaled.

How these and other requirements are achieved is covered in detail in the individual control statement descriptions. However, the following gives you an idea where, in the context of a simple control file, these requirements are specified.

You will find PRINTU easy to understand and use if for the first few times you use it you confine yourself to the four required control statements. A useful report can be generated with these four alone. The four control statements and their characteristics are:

IDENT

This statement provides a means of assigning a name to the control file. It has no effect on input data selection, format, or content of the printed report.

OUTPUT

This statement serves only to assign a unique name to the resulting print report. This statement is optional in the sense that, if not included, a prompt will request the name at program run time.

INPUT

INPUT is one of the two most important field definition statements. It is here that you:

- Select the input data file.

If the INPUT statement is omitted, this information will be requested via a prompt at run time.

- Describe the fields that you need from within the input data record.
- Select break fields.

- Request a new printed page concurrent with a break field change.
- Select whether you print only totals or intermediate values as well.
- Request to read an ISAM input file sequentially.

PRINT

PRINT is the other field definition statement that is of prime importance. It is here that you:

- Make the logical connection between fields identified in the input statement and fields to be printed.
- Assign column headings for each field printed.
- Select numeric fields that are to be accumulated.
- Format the numeric data to be printed.
- Assign column spacing.

All nine control file statements are discussed in the following subsections. Optional input for each statement is shown in brackets. Comment lines are allowed only as shown for each entry. An example is given for each statement as it is presented. Each example is part of an overall exercise to build a usable control file; therefore, some examples build on concepts presented with other statements. The completed control file is shown as part of the example in Section 9.3.1.

COMPUTE (Optional)

9.2.1 COMPUTE (Optional)

COMPUTE is a multiline control statement that allows you to include data in your report that does not exist in the input data file. This new data is a result of a mathematical process performed on the input data. The COMPUTE line is followed by one or more additional lines describing the mathematical process. There can be up to eight of these additional lines. The multiline format is:

```
COMPUTE  
[;comment]  
name = (expression)[-n]
```

where:

name is the symbolic name given to the computation result and is used for reference by the PRINT statement (see Section 9.2.9). It must neither duplicate any input field name nor any previous computation result name.

(expression) is any valid DIBOL expression. It must be enclosed in parentheses as shown and can be an alpha or decimal field or a literal. See SPECIAL RULES below.

-n is an integer specifying the number of decimal places the resulting computation will have. See SPECIAL RULES below.

The comment line is a separate line as shown.

SPECIAL RULES FOR COMPUTE:

These special rules concern the number of decimal places for operands within a mathematical operation. The PRINT control statement, unlike the DIBOL language, maintains the number of decimal places. In all arithmetic operations, the operands must have an equal number of decimal places. Operands can be either constants or variables. The rules listed below determine the resulting decimal places:

Operation	Effect
addition (+)	no change in decimal places
subtraction (-)	no change in decimal places
multiplication (*)	(*) the sum of decimal places
division (/)	the difference in decimal places

To adjust the number of decimal places of any operand, multiply by a suitable decimal representation for the constant 1. For example, if operand 1 has two places, and operand 2 has no places; then operand 1 * operand 2 * 1.00 will be acceptable and will result in a product which will have four decimal places. For example:

```
COMPUTE  
;compute total pay  
TPAY=(CRATE*HOURS*1.00)-2
```

Total pay (TPAY) is equal to compensation rate (CRATE) multiplied by hours worked (HOURS). TPAY is to have two decimal places.

END (Optional)

9.2.2 END (Optional)

The END statement is optional and consists of a line containing only the statement:

END

EXECUTE (Optional)

9.2.3 EXECUTE (Optional)

EXECUTE creates the DIBOL stop and chain exit in the report program. The format is:

```
EXECUTE filespec                               [;comment]
```

where:

filespec is the file specification of the program which PRINTU will chain to upon completion of the report.

Example:

```
EXECUTE DL1:STATUS.TSD
```

Upon completion of the report, the report program would chain to a program called STATUS.TSD on device DL1:.

NOTE

The EXECUTE statement must appear after IDENT and before the INPUT statements.

HEAD1 and HEAD2 (Optional)

9.2.4 HEAD1 and HEAD2 (Optional)

HEAD1 defines the first heading line appearing on each page of the report and HEAD2 defines the second. HEAD2 is optional. Both headings are centered on the page. The form is:

```
HEAD1 'text'                [;comment]
HEAD2 'text'                [;comment]
```

where:

text is a string of characters from the DIBOL set, exclusive of quotes, up to 132 characters long.

There may be more than one HEAD1 or HEAD2 line. If so, the individual text lines for given HEADn statements are concatenated. This is necessary for long titles. If HEAD1 is less than 67 characters, the line will be expanded by inserting a space between each character. The total number of characters for each line is limited to 132.

Example:

```
HEAD1 'PAY '
HEAD1 'ACCOUNTABILITY'
HEAD2 'BY EMPLOYEE AND TASK CODE'
```

The heading on each page would then appear as follows:

```
PAY ACCOUNTABILITY
BY EMPLOYEE AND TASK CODE
```


IDENT

9.2.5 IDENT

IDENT is the first entry in the control file. It provides the control file title and appears on the first line of the DIBOL listing of the report program and at the top of each printed report page.

It is of the form:

IDENT program, author [;comment]

where:

program is the identifier you want to assign to the report. It can be up to 22 characters long with a slash (/) used as a word separator. Hyphens are not accepted.

author is any text up to 24 characters long. Specification of author is required.

Example:

IDENT PAY/ANALYSIS, Your name

The START line of the DIBOL listing would look like the following:

START ;PAY ANALYSIS -YOUR NAME

INDEX/LIST (Optional)

9.2.6 INDEX/LIST (Optional)

INDEX and LIST are statements that enable PRINTU to utilize a tag file generated by either the SORT or SORTGEN utility. See the TAGS (LIST, INDEX) qualifier for SORT (Chapter 12) and the TAGS:LIST and TAGS:INDEX options for SORTGEN (Chapter 11). The usefulness of such a tag file becomes apparent when you consider that it is unlikely that the input data file will be arranged in the order that you want for your report.

There are two solutions to this problem. You can either reorder the input data file, which is especially time-consuming if you want several kinds of reports from the same data, or you can use the sort-generated tag file. The tag file allows access to the input data file in its original form. Consequently, it is not necessary to reorder the entire input file when a different sort sequence is desired. All you have to do is generate a new tag file.

A tag sort is first carried out on the input data file. The output is a file of records containing only the relative record number, or if it is an ISAM input file, the file contains the sort keys also.

The tag file is specified for use in PRINTU by either the INDEX or LIST statement. When INDEX is used, it implies that the input data file (as defined by the INPUT statement) is an ISAM file. When LIST is used, it implies that the input data file (as defined by the INPUT statement) is a sequential file.

If either INDEX or LIST is used, it must precede the INPUT statement in the control file.

The format, if the input file is a sequential file, is:

```
LIST filespec                               [;comment]
```

where:

filespec is the file specification of the tag file generated by the SORT or SORTGEN utility.

There are no qualifiers with LIST. The tag file is assumed to consist of records which are seven-digit decimal numbers.

Example:

```
LIST DK1:SORTO.DDF
```

This statement line specifies tag file SORTO.DDF on DK1:. SORTO.DDF is a result of previously having run a sort with the TAGS:LIST option on the sequential file defined in the PRINTU control file INPUT statement. This tag file is a file of seven-digit numbers.

The format, if the input file is an ISAM file, is:

```
INDEX filespec/recl,key,length,name        [;comment]
```

where:

- filespec is the file specification of the tag file generated by the SORT or SORTGEN utility.
- recl is an integer defining the record length of the tag file as generated by the SORT or SORTGEN utility.
- key is an integer defining the starting position of the ISAM key within a tag file record.
- length is an integer defining the length of the ISAM key within a tag file record.
- name is the name of the field containing the ISAM key. This field must also be identified in an INPUT field description line.

Example:

```
INDEX RK1:SRTOUT.DDF/12,8,5,KEY
```

This statement line specifies a secondary input file SRTOUT.DDF on RK1:. SRTOUT.DDF is the result of previously having run a sort with the TAGS:INDEX option on the ISAM file defined in the PRINTU control file INPUT statement. This example indicates that the secondary input file record length is twelve characters long, and that the ISAM key (named KEY) is five characters long, beginning at character position eight. The seven-character relative record number is not used.

If neither the INDEX nor LIST statement is included in the control file, the input file is processed as a sequential file.

INPUT

9.2.7 INPUT

INPUT is a multi-line control statement. It specifies the data source for the report program (Input Statement Line) and describes the input data file's record structure (Field Description Lines).

NOTE

The input file must be sorted in relation to the break field(s) before it is supplied to the report program. This requirement becomes obvious when you become familiar with the function of a break field.

The multiline format is:

```
INPUT [filespec][ /SU ][ /IS ]
[;comment]
[name],  $\left. \begin{array}{c} A \\ D \end{array} \right\} n$  [.m] [,Lr [P]]
[;comment]
```

The Input Statement Line and Field Description Lines are discussed separately.

Input Statement Line

Taking the INPUT statement by itself, the form is:

```
INPUT [filespec][ /SU ][ /IS ]
[;comment]
```

where:

- filespec** is the file specification of the input data file from which the report is generated.
- /SU** is the summary switch. When included, it causes suppression (during print) of individual values being accumulated. See PRINT, Section 9.2.9.
- /IS** indicates that the input data file is an ISAM file to be read sequentially. IS must be used when you want to generate a report directly from an ISAM file without using the INDEX statement. See Section 9.2.6.

NOTE

The default condition assumes that the input data file is a sequential file. If the input data file is an ISAM file, either the IS option or the INDEX statement must be used.

Comments are on a separate line as shown.

If the file specification alone is omitted, the report program will request it at run time. The message will be:

INFILE:

The options SU and IS may not be supplied at run time, but can be selected in the control file, with a file specification to be supplied at run time, by using the form:

INPUT /SU

The ability to supply the input data file specification at run time, combined with the same ability for the output file, makes it possible to use a given report program with any number of input data files and to produce a separate output for each. Of course, the record structure must be the same in all such input files.

Field Description Lines

Before analyzing a field description line, it is necessary to understand exactly what a field description line does.

One function of a field description line is to describe the input data file's record structure. The data record is partitioned into fields, and the nature of each field is described in detail. These details include name, data type, size, and, for decimal fields, number of decimal places.

The field description line is also used to identify break fields. The break field contains data that is constant over a series of records. However, for each break field there are corresponding fields which may contain data which can vary from record to record. It is this relationship that is the basis for reports. And, in PRINTU, it also controls when data is printed.

An example may help to clarify this concept:

A record might contain an employee's name, the date, a task code, and the hours worked for this task code. For a given employee, there will exist at least one record for every task code for which time was charged. In the simplest case, you may want a report showing the hours worked by each employee. Here the field containing the employee's name would be identified as the break field. The report would then print all the hours worked by each employee. Note that it may also be desirable to show, for a given employee, the total hours worked in a given day or on a given task.

More than one break field can be identified. For example, it may also be desirable to report hours worked for each task code, as well as for each employee. In this case, task code would also have to be identified as a break field. With more than one break field, a way to show this relative importance is necessary. PRINTU allows you to identify both break fields within the control file INPUT statement and to show their relative importance. In this second case, task code hours would appear as subtotals every time the task code changed, and a total of hours by each employee's name would appear as each name changed.

From the preceding paragraphs, it can be seen that a break field controls the printing of data. Whether it is a summary print or a detail print is also determined by the INPUT statement. Since the file is read sequentially, the input file must, in the first case above, be sorted by employee name. If other break fields of lesser importance were identified, they also would have had to have been identified as minor keys (in the correct order) in the sort. Task code as a break field would be identified as a minor key in the sort.

Again, the format for a field description line is:

[name], $\left. \begin{array}{c} A \\ D \end{array} \right| n [.m] [,Lr [P]]$
[:comment]

where:

name is the symbolic name of the field to be used for reference by the COMPUTE and PRINT statements. See Sections 9.2.1 and 9.2.9. It can contain up to six alpha characters.

$\left. \begin{array}{c} A \\ D \end{array} \right|$ indicates alpha or decimal field type.

n is the size of the field expressed in decimal characters.

m is the number of decimal places in the field and is valid only for decimal fields.

Lr indicates a break field in which the r is a single decimal digit, which expresses the relative importance of the break field compared to other break fields (1 is the least important and 9 is the most important). When a "break" occurs, the total is printed for fields being accumulated. This total also includes all other fields of lower break level. See PRINT, Section 9.2.9. All accumulated values are then cleared to zero.

P indicates that the report will start a new page after the totals for this break are printed.

Comments are on a separate line as shown.

Twenty is the maximum number of labeled fields allowable within the INPUT control statement. All the fields must be defined and these fields must all be in the same record.

NOTE

A field named in the INDEX statement (see Section 9.2.6) must also be identified by an INPUT field description line.

Example (INPUT statement and field definition lines):

```
INPUT EXAMP1
;comment line
NAME, A15,L2P
, A7
DATE, D6
TCODE, A4,L1
CRATE, D3.2
HOURS, D2.1
```

Five fields have been identified for the data contained in each record of file EXAMP1. Data areas within the record that are not of interest can be ignored by using an unnamed alpha field of appropriate size. In this example, there is one such area seven characters long. The two fields, NAME and TCODE, are identified as break fields and NAME is of greater importance. Whenever the data in TCODE changes, the output lists specified totals (and individual values, unless a summary is requested). The exact format of the data to be printed will be defined in the PRINT statement (see Section 9.2.9). When NAME changes, the totals printed include the totals from intermediate TCODE lines. A new page is started after each NAME change. HOURS is a two-character field with one decimal place. CRATE is a three-character field with two decimal places.

OUTPUT

9.2.8 OUTPUT

OUTPUT identifies, by nature of your response, either a printer or a report file that is the destination of the output that results from running the report program. Its form is:

OUTPUT filespec

where:

filespec is the file specification of the output report.

No comment line is allowed with this statement.

Example:

OUTPUT RK1:PAYACC.DDF

When the report program is run, the resulting report is placed on RK1: and is named PAYACC.DDF.

NOTE

The OUTPUT statement must appear after IDENT and before the INPUT statement.

PRINT

9.2.9 PRINT

PRINT is a multiline control statement that defines the format of the final print report and identifies the variable fields that are to be accumulated (added) and printed as totals for a given break field. The PRINT control statement is followed by one or more additional statements describing the overall print format and accumulation fields. The multiline format is:

```
PRINT  
[;comment]  
name,'text' [,A] [,picture]  
or  
,An
```

where:

- name** is the name of a field from an INPUT field description line (see Section 9.2.7) or the name assigned to a mathematical result within a COMPUTE statement (see Section 9.2.1).
- text** is a string of characters from the DIBOL character set (excluding single quotes). This text is used as the heading for the individual columns for the data fields identified by name as well as in the line printed for totals for each break field. An asterisk in the text string will cause the remaining text to be printed on the next line. Text is automatically centered. An asterisk is treated as a space when the text is printed.
- A** indicates that the field is to be accumulated and the sum is to be printed as a total for a break field. This field must be decimal. If the /SU option were selected in the INPUT statement, only the total would be printed (summary print), not the intermediate values.
- picture** is a string of text in the form of a DIBOL data format field. Here it is valid only for decimal fields. If a picture is not included, PRINT will create one using the description of the field. If it is an accumulated field, two extra places will be assumed. The form is:

XX,XXX.XX-
- ,An** is an alternate format line that will produce blank column separators of n characters in width between printed output columns (where n is an integer from 1 to 9). If not included, two blank columns will be assumed.

The comment line is a separate line as shown.

Example:

```
PRINT  
;comment  
NAME, 'EMPLOYEE'  
,A4  
TCODE, 'TASK*CODE'  
,A4  
HOURS, 'HOURS*WORKED',A,ZZ.Z  
,A3  
TPAY, 'TOTAL*PAY',A,ZZZZ.XX
```

Assuming that INPUT and COMPUTE statements remain as previously illustrated, in this example the print output would be:

EMPLOYEE	TASK CODE	HOURS WORKED	TOTAL PAY	
name1	code1	ZZ.Z	ZZZZ.XX	
		ZZ.Z	ZZZZ.XX	
TASK CODE TOTAL		ZZZZ.Z	ZZZZZZ.XX	
name1	code2	ZZ.Z	ZZZZ.XX	
		ZZ.Z	ZZZZ.XX	
		ZZ.Z	ZZZZ.XX	
TASK CODE TOTAL		ZZZZ.Z	ZZZZZZ.XX	
EMPLOYEE TOTAL		ZZZZ.Z	ZZZZZZ.XX	
name2	code1	ZZ.Z	ZZZZ.XX	(New Page)
(and so forth)				

9.3 PRODUCING THE REPORT PROGRAM

The first step in using PRINTU is to create the control file with an editor. The control file simply consists of the required PRINTU statements from Section 9.2 plus any optional statements desired. Regardless of which editor you use to create your PRINTU control file, you will be required to specify a name for the file resulting from your edit session. Specify this name as input to the PRINTU program. The resulting DIBOL program source file is compiled and linked just like any other such program. The following dialog illustrates in detail the steps required. The same control file statements developed for the examples in Section 9.2 are used.

There are five steps:

1. Create the control file:

```
.R DKED
*PTSAMP.TXT =
IDENT PAY/ANALYSIS, YOUR NAME
HEAD1 'PAY ACCOUNTABILITY'
HEAD2 'BY EMPLOYEE AND TASK CODE'
EXECUTE URPROG.TSD
;return to your
;program when
;finished
OUTPUT RK1:PAYACC.DDF
INPUT EXAMP1.DDF
;input file is a sequential file on the default device, no
;summary wanted
NAME, A15,L2P
,A7
DATE, D6
TCODE, A4,L1
CRATE, D3.2
HOURS, D2.1
COMPUTE
;compute total pay
TPAY = (CRATE*HOURS*1.00)-2
PRINT
;comment
NAME, 'EMPLOYEE'
,A4
TCODE, 'TASK*CODE'
,A4
HOURS, 'HOURS*WORKED',A,ZZ.Z
,A3
TPAY, 'TOTAL*PAY',A,$$$$.XX
END
<GOLD/COMMAND>
EXIT
```

2. Run PRINTU:

Enter the following command:

```
.R(FU) PRINTU
```

The general form for entry of arguments in PRINTU in response to the asterisk prompt is:

```
*outfil,lstfil = txtfil
```

where:

outfil is the file specification to be assigned to the output of PRINTU. It consists of the appropriate device, the name of the text file containing the PRINTU control file, and a default .DBL extension since the output of PRINTU is supplied to the compiler.

lstfil is the file specification of the PRINTU list file.

txtfil is the name given to the PRINTU control file in the edit session. There is no default extension.

For the example being developed here the entry is:

```
*PTSAMP.DBL,LP: = PTSAMP.TXT
```

which produces the DIBOL source file PTSAMP.DBL on the default device and a listing of this program on the line printer.

It might be useful to first find the syntax errors, using the following command:

```
*PTSAMP.DBL,TT: = PTSAMP.TXT
```

edit out the errors, and (assuming you do not need a listing) enter:

```
*PTSAMP = PTSAMP.TXT
```

3. Compile:

```
.R DICOMP PTSAMP = PTSAMP
```

4. Link:

```
.LINK PTSAMP,ODIBOL,DIBOL
```

5. Run the report program:

```
.R PTSAMP
```

NOTE

Ensure that the input file is sorted in relation to the chosen break fields before the report program is run. See Section 9.2.7, INPUT.

PTSAMP is the report program, and the result of running it will be file PAYACC.DDF which is written to RK1: in the format specified in the PRINT statement.

CHAPTER 10

ISAM (ISMUTL)

10.1 INTRODUCTION

ISMUTL is the DIBOL utility program used to build and reorganize CTS-300 ISAM (Indexed Sequential Access Method) files. This chapter explains each of the various functions of ISMUTL, using flowcharts (where appropriate), actual program dialog, and examples.

There are a number of interrelated considerations that must be dealt with if you are to construct an efficient ISAM file. If you are not familiar with ISAM files, see Appendix A for a discussion of ISAM file concepts and characteristics.

Section 5.4.5.1 in Chapter 5 discusses precautions that should be taken when multiple users are accessing the same ISAM file.

10.2 USING ISMUTL

ISMUTL is the DIBOL utility that allows you to select the parameters to define, build, and support a desired ISAM file. ISMUTL performs four selectable operations:

- An ISAM file is constructed by using the create (CREATE) feature of ISMUTL.
- The status (STATUS) feature of ISMUTL allows the operator to check file structure, file status, utilization of overflow, and (in chain mode) append areas. When one or both of the latter become full, or nearly so, it is usually necessary to rebuild your ISAM file. This may be done with either the create or the reorganization feature of ISMUTL.
- Assuming the basic parameters are still valid, the reorganization (REORG) feature of ISMUTL can automatically create a new ISAM file which restores file expansion areas to their original values and consolidates all the data. The result is then a larger (or perhaps smaller) addressable (indexed) area.
- The exit (EXIT) feature terminates ISMUTL and returns control to the run-time system (TSD or XMTSD) or the operating system.

10.2.1 ISMUTL Requirements

Before you actually use any of the features of ISMUTL, there are some requirements and characteristics of the utility that must be understood.

10.2.1.1 Single-User Operation — In order to execute a single-user DIBOL program that contains ISAM access statements, ISAM must first be selected during CTSGEN.

10.2.1.2 Time-Shared Operation — Time-shared operation with ISAM and ISMUTL requires special attention during CTSGEN. The CTSGEN must include ISAM, and a specific number of channels must be specified in CTSGEN, depending on the number of logical volumes to be assigned in ISMUTL CREATE. It is also possible to add a data file during REORG, and this would affect channel requirements.

Twelve is the maximum number of channels required if you are creating or reorganizing an ISAM file of seven data volumes. Time-shared system channel requirements, as a function of data volumes, are listed below:

NUMBER OF LOGICAL DATA VOLUMES	NUMBER OF CHANNELS REQUIRED
1	6
2	7
3	8
4	9
5	10
6	11
7	12

NOTE

Under time-sharing, no other user should access the ISAM file while it is being created or reorganized.

10.2.1.3 Error Messages — Error messages associated with ISMUTL are documented in the *CTS-300 System Message Manual*.

10.2.2 Running ISMUTL

Assuming proper linking and a CTSGEN that includes ISAM, the command to execute ISMUTL (ISMUTL.SAV) for a single-user system is:

```
.R ISMUTL
```

or, with the same assumptions, plus sufficient channels, the command to execute ISMUTL (ISMUTL.TSD) for a time-shared system is:

```
*R ISMUTL
```

The response is:

```
CTS300 ISAM UTILITY PROGRAM, Vnn-nn  
SELECT FUNCTION (CREATE,STATUS,REORGANIZE, OR EXIT):
```

Your response to this selection message is the first letter of the chosen function. But first see the appropriate section below for the operating details of that function.

10.2.3 Creating a File (CREATE)

This section contains a discussion of the special characteristics of CREATE and details of the CREATE process.

10.2.3.1 Special CREATE Characteristics

Input File

Input for ISAM file creation may be a sequential file; another ISAM file; or an ISAM file may be created without an input file. If the input is a sequential file, there is a special requirement: the file must first be sorted in ascending order by the same key that will become the ISAM key.

After naming the input file, you will be asked if you want to delete the input file after CREATE. If you choose to delete the input file, CREATE opens two checkpoint files which, in an alternating manner, keep track of the cleanup process associated with the deletion of a same-name file.

If a machine malfunction occurs during CREATE with the input file deleted, the checkpoint files allow resumption at the correct point.

See Section 10.2.3.5 for details on restarting the process.

If you decide to delete your input file, you may want to make a back-up copy to eliminate loss of data due to a hardware malfunction.

NOTE

Certain ASCII characters must be avoided in ISAM files. These characters (listed below) interact with intermediate work files during ISMUTL CREATE or later REORG. Any of these characters will cause truncation of the record in which it is found.

Character	Decimal Code
Null	0
HT	9
LF	10
VT	11
FF	12
CR	13
Z	26
Esc	27

Output File

You must name the resulting ISAM file at the beginning of the CREATE dialog. Allocation of the index and data files to the system resources (devices) takes place at the end of the dialog.

The output file name can be any six-character file name. The default extension is .ISM, which is assigned to the index file. The data files are assigned default extensions of .IS1 through .IS7. If an extension is specified, it must be at least two characters long; if it is three characters, the last character cannot include the numbers 1 through 7. These numbers are assigned by ISMUTL and will override any third character position. The extension .ROG cannot be used either, since that is a temporary extension used by the system when the file is reorganized. If the output file name is the same as the input file name, temporary files, .TMP and .TM1 through .TM7, are built by CREATE and there should be no other files of that name on the system.

Device identification takes place at the end of the CREATE dialog when files are allocated.

There must be enough contiguous space on each device to open the associated data files and, for one device, to open the index file as well. If any difficulty arises, see Section 10.2.3.5.

Ordinarily volumes containing data files must always be mounted for use on the same devices and units as those on which they were created. However, by using the RT-11 ASSIGN command for device names, you are not restricted to the same physical devices.

Total File Requirements

In addition to files already mentioned, in all CREATE operations there are also two work files that are opened to build the output index file. CREATE will try to find space for the work files on the device you specify for your output files. If enough space is not found, CREATE will ask for additional devices. These are temporary files that will be deleted automatically after CREATE has built the index file.

Below is a summary of the files that will be automatically created, depending on deletion and same-name choices:

If you want to delete the input file and both input and output files have the same name:

filnam.CK1	(checkpoint file on system device)
filnam.CK2	(checkpoint file on system device)
filnam.TMP	(temporary file)
filnam.TM1	to .TM7 (depending on the number of input data files)
filnam.ROG	(temporary REORG file)
filnam.RO1	to .RO7 (depending on the number of input data files)
filnam.WK0	(work file)
filnam.WK1	(work file)

If you want only to delete the input file:

filnam.CK1	(checkpoint file on system device)
filnam.CK2	(checkpoint file on system device)
filnam.WK0	(work file)
filnam.WK1	(work file)

If you do not want to delete the input file:

filnam.WK0	(work file)
filnam.WK1	(work file)

Auto-Create

The CREATE function includes the optional capability of operating from an external data file of input responses. This is a special function for experienced users and should not, in any case, be invoked to create a file with unproven parameters. For this reason, a complete discussion of this capability is provided under Section 10.2.7.2, rather than here.

10.2.3.2 Design/CREATE Process — Creating an ISAM file is a five-step process. The first four of these steps are preliminary to running CREATE:

1. Determine the key/record/group/block size relationship.
2. Determine record addition characteristics for load exclusion, overflow, and append area requirements.
3. Determine the output names.
4. To satisfy requirements which apply to your proposed ISAM file, determine the following in advance of running CREATE:
 - Whether to do an auto-create
 - Input file (name, ext .DDF assumed)
 - Whether to delete your input file (ISAM or sequential only)
 - Output file name
 - Approximate number of input records (sequential only)
 - Record length (only if no input file)
 - Number of records per group
 - Load exclusion value
 - Whether to fill out the group to end of block boundary
 - Append area size
 - Overflow area size

The following two factors are automatically supplied if the input file is an ISAM file:

- Key length
- Key location
- Whether to allow duplicate keys, and, if so, whether to place at beginning or end (if ISAM input file, question appears only if not already allowed).

- Output allocation:

Index File

Data Files

The device is specified without the colon.

5. Select CREATE by typing C in response to the ISMUTL selection message and enter the values you have just determined. The following section will help you during the create process.

10.2.3.3 CREATE Dialog — This section contains the actual text for the create process. All create questions are included here, even though no given create could include every question. Allowable ranges are shown as a final check on your answer. Figure 10-1 illustrates the dialog flow for the possible input file types and subsequent choices.

The following is the CREATE dialog:

DO YOU WANT TO DO AN AUTO-CREATE? (YES/NO)

Answer NO, or carriage return, unless you are familiar with the auto-create process. The special procedures for this function are detailed in Section 10.2.7.2.

INPUT FILE (DEV:NAME.EXT):

If not specified, assumed device is the default device and assumed extension is .DDF. A carriage return implies no input file, in which case the next question does not appear.

DELETE INPUT FILE AFTER CREATE (YES/NO):

If yes, your input file will be automatically deleted after the output file is created.

OUTPUT FILE (NAME.EXT, DEFAULT EXT = ISM):

Enter just the file name; device names are supplied later.

APPROXIMATE NUMBER OF INPUT RECORDS (0-16,777,215):

Appears only if sequential input is specified. You can specify up to the system limit. Unneeded space is not used. Your answer determines the size of the work files.

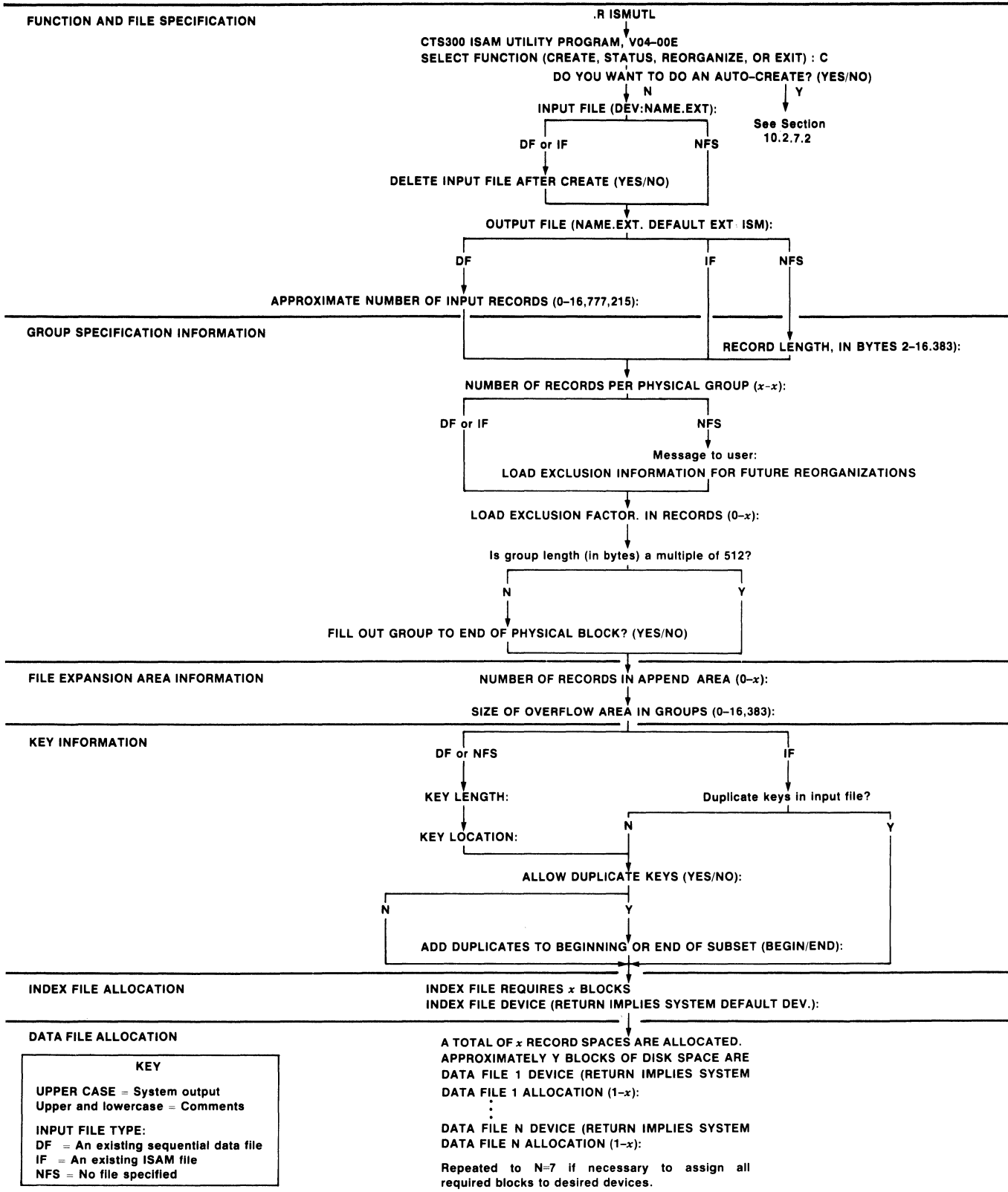


Figure 10-1 ISAM CREATE Flowchart

RECORD LENGTH, IN BYTES (2-16,383):

Appears only if no input is specified.

NUMBER OF RECORDS PER PHYSICAL GROUP (x-x):

The range is determined by a minimum group size of 132 bytes and by a maximum group size of 127 records or 16,383 bytes, whichever is smaller.

LOAD EXCLUSION INFORMATION FOR FUTURE REORGANIZATION

Appears only if there is no specified input file. Since there is no initial data, there can be no exclusion. However, when the file is reorganized, the amount of exclusion selected here will be provided.

LOAD EXCLUSION FACTOR, IN RECORDS (0-x):

The maximum number of records that can be allocated for exclusion is always one less than the total number of records in a group. If there was no input file specified, this message would be preceded by the following:

FILL OUT GROUP TO END OF PHYSICAL BLOCK? (YES/NO)

This answer depends on the record/group/block size relationships and is a matter of judgement. You trade efficiency for access speed.

NUMBER OF RECORDS IN APPEND AREA (0-x):

The append area is for the addition of records with key values greater than the initial input. Although the low range of your choice is labeled 0, you actually get at least one group. The maximum number is limited by system storage capacity.

SIZE OF OVERFLOW AREA IN GROUPS (0-16,383):

The overflow area is for addition of records of intermediate key value to groups that may eventually be filled.

KEY LENGTH

Key length is the length of the key in bytes. This question appears only if your input is not an ISAM file.

KEY LOCATION

Key location is the beginning byte position of the key within the record. The first byte position of a record is number one. This question appears only if your input is not an ISAM file.

ALLOW DUPLICATE KEYS (YES/NO):

Do you want to allow more than one record with the same key value? This question appears only if your input is not an ISAM file, or if your input ISAM file did not allow duplicate keys.

ADD DUPLICATE KEYS TO BEGINNING OR END OF SUBSET (BEGIN/END):

To place added records before or after existing records with the same key, respond with B or E. Storage at the beginning means that the last record will be stored at the beginning of the duplicate set and will be the first record accessed with a READ. Storage at the end means that the last record will be stored at the end of the duplicate set and will be the last record accessed with a READS. This question is not asked if duplicates are not allowed.

INDEX FILE ALLOCATION.

INDEX FILE REQUIRES x BLOCKS.

INDEX FILE DEVICE (RETURN IMPLIES SYSTEM DEFAULT DEV.):

The program has calculated the space requirement for the index file. The x indicates the blocks necessary to create the index file. Respond with the device you want without the colon. A carriage return indicates the default device. The entire index file must be on a single device.

DATA FILE ALLOCATION.

A TOTAL OF x RECORD SPACES ARE ALLOCATED.

APPROXIMATELY y BLOCKS OF DISK SPACE ARE REQUIRED.

DATA FILE 1 DEVICE (RETURN IMPLIES SYSTEM DEFAULT DEVICE):

At this point you must finalize your decision on how many files (logical volumes) the ISAM file will have and upon what device(s) they will be placed. The values of x and y are approximations, but they are close enough to indicate space requirements. With a small file, it may be possible to place the entire data file on the same device as the index file. Select the device, without the colon, for the first data file.

DATA FILE 1 ALLOCATION (1-x):

Assign the blocks to the first device. The range is from one to the maximum required. Select a value in keeping with the disk capacity. If a choice is lower than the upper limit, unassigned blocks will have to be assigned to other data volumes. The above sequence of assignment of device and blocks will continue as follows:

NOTE

A choice higher than, or equal to, the upper limit will default to the upper limit, thereby terminating the dialog.

DATA FILE 2 (through 7) DEVICE (RETURN IMPLIES SYSTEM DEFAULT DEVICE):

Assign device two (through seven).

DATA FILE 2 ALLOCATION (1-n):

Assign the blocks to the second (through seventh) device. There will be a diminishing number of blocks to be assigned, as indicated by the value of n.

If the end of the sequence is reached but not all the blocks have been assigned, the process is repeated, starting at the beginning of the data file assignment sequence.

It is recommended that you keep an ISAM file on as few physical and logical volumes as possible because nothing is gained unless you have a fragmented disk due to bad blocks.

If any difficulty is encountered, see Section 10.2.3.5.

10.2.3.4 CREATE Example — The following is an example of the dialog for the creation of an ISAM file from a sequential input file. The input file is known to consist of 72-byte records with the key of eight bytes located at the beginning of the record. Seven records per group will be chosen to allow a group to exist within a 512 byte block. Since this leaves little wasted space, fill-out to the end of the block is chosen. There are somewhat less than 250 records in the input file.

New records to be stored are expected to be scattered throughout the file and, in addition, there will be new records whose key values are greater than any presently existing in the input file. A load exclusion factor of two is chosen to handle the initial scattered records of intermediate key value. There are most likely less than 700 new intermediate records to be added that cannot be accommodated by load exclusion; however, an overflow area of 100 groups is chosen. There are estimated to be somewhat less than 200 records that will be added to this file that have greater than present key values. A value of 200 records is therefore chosen for the append area size. Duplicate keys will not be allowed.

After file allocation is complete, ISMUTL automatically gives you the status of the new file. Notice the values. Later, in Section 10.2.4.1, STATUS Example, this same file with added records will be shown.

The CREATE Example:

```
.R ISMUTL
```

```
CTS300 ISAM UTILITY PROGRAM, Vnn-nn
```

```
SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT): C
```

```
DO YOU WISH TO DO AN AUTO - CREATE? N
```

```
INPUT FILE (DEV:NAME.EXT): XDATA1.DDF
```

```
DELETE INPUT FILE AFTER CREATE (YES/NO): N
```

```
OUTPUT FILE (NAME.EXT, DEFAULT EXT = ISM): XDATA1.ISM
```

```
APPROXIMATE NUMBER OF INPUT RECORDS (0-16,777,215): 250
```

```
GROUP SPECIFICATION INFORMATION.
```

```
    NUMBER OF RECORDS PER PHYSICAL GROUP (2-127): 7
```

```
    LOAD EXCLUSION FACTOR, IN RECORDS (0-6): 2
```

```
    FILL OUT GROUP TO END OF PHYSICAL BLOCK? (YES/NO) Y
```

```
FILE EXPANSION AREA INFORMATION.
```

```
    NUMBER OF RECORDS IN APPEND AREA (0-16,776,863): 200
```

```
    SIZE OF OVERFLOW AREA IN GROUPS (0-16383): 100
```

```
KEY INFORMATION.
```

```
    KEY LENGTH: 8
```

```
    KEY LOCATION: 1
```

```
    ALLOW DUPLICATE KEYS (YES/NO): N
```

```
INDEX FILE ALLOCATION.
```

```
    INDEX FILE REQUIRES 107 BLOCKS.
```

```
    INDEX FILE DEVICE (RETURN IMPLIES SYSTEM DEFAULT DEV.)
```

DATA FILE ALLOCATION.

A TOTAL OF 553 RECORD SPACES ARE ALLOCATED.
APPROXIMATELY 80 BLOCKS OF DISK SPACE ARE REQUIRED.
DATA FILE 1 DEVICE (RETURN IMPLIES SYSTEM DEFAULT DEVICE)
DATA FILE 1 ALLOCATION (2-80): 80

10-MAY-83	FILE LOCATION	GROUPS ALLOCATED
KEY LENGTH IS- 8	INDEX FILE DK	104
RECORD LENGTH IS- 72	DATA FILE -1 DK	78
LEVELS OF INDEXING IS- 1		
LOAD EXCLUSION FACTOR- 2		
KEY STARTS AT LOCATION- 1		
DUPLICATE KEYS ALLOWED- NONE		
CURRENT NUMBER OF RECORDS- 238		
GROUPS ALLOCATED TO OVERFLOW- 100		
GROUPS REMAINING IN OVERFLOW- 100		
NUMBER OF RECORDS PER GROUP- 7	GROUP LENGTH IS- 512	
SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT): E		

END ISAM UTILITY.

Note that the ISAM file created in the example could become an inefficient file because such a large overflow area was chosen. Because of this large overflow area, the index file is larger than the one data file.

10.2.3.5 Handling CREATE Problems

Failure During Clean-up

If a machine malfunction occurs during the cleanup routine of a CREATE with input file deleted, it is possible to complete the operation. To do so, you must have at least one of the checkpoint files on the system device. Start the CREATE again and specify the same file previously named in response to the input file question. The program searches for a file with the CK1 or CK2 extension and, when found, restarts and completes the function automatically.

NOTE

Remember that if CREATE does not terminate normally, do not delete temporary files, work files, or checkpoint files. Doing so may result in the loss of your input ISAM file.

More Space Required for Output Files

If, at the end of the CREATE dialog, you have failed to allocate all the space required by the ISAM file, you will receive a message:

ALL FILES ALLOCATED AND MORE SPACE IS REQUIRED. PLEASE TRY AGAIN.

The allocation sequence starts again, and you must allocate all of the required file space before you reach the end.

Insufficient Work File Space

It is possible that there is not enough space for the CREATE function to open all the files necessary to operate. If this is the case, the following message will appear:

WORK FILES REQUIRE x BLOCKS OF DISK SPACE WORK FILE DEVICE:

Respond with a device name.

10.2.4 Determining the Status of an ISAM File (STATUS)

10.2.4.1 STATUS Selection and Characteristics — To determine the status of an ISAM file, type S in response to the ISMUTL selection message. The program responds with a description of group structure, of current number of records, and of how much overflow area has been used. Status selected in this manner does not provide any information on the append area space which remains. To obtain information on the free append space, see Section 10.2.7.1.

10.2.4.2 STATUS Example — The ISAM file created in the example in Section 10.2.3.4 is used to illustrate the changed STATUS information after the addition of some records.

The number of records has increased by 110 records to a total of 348 records. There are now 12 overflow groups in use for a maximum of 84 records in overflow (12 x 7 records/group); therefore, the remaining added records are probably accounted for by load exclusion record spaces becoming filled and by records going into the append area.

This same file is reorganized in Section 10.2.5.3.

The STATUS Example:

```
.R ISMUTL
```

```
CTS300 ISAM UTILITY PROGRAM, Vnn-nn
```

```
SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT): S
```

```
INPUT FILE (DEV:NAME.EXT): XDATA1.ISM
```

```
10-MAY-83
```

```
KEY LENGTH IS- 8
```

```
RECORD LENGTH IS- 72
```

```
LEVELS OF INDEXING IS- 1
```

```
LOAD EXCLUSION FACTOR- 2
```

```
KEY STARTS AT LOCATION- 1
```

```
DUPLICATE KEYS ALLOWED- NONE
```

```
CURRENT NUMBER OF RECORDS- 348
```

```
GROUPS ALLOCATED TO OVERFLOW- 100
```

```
GROUPS REMAINING IN OVERFLOW- 88
```

```
NUMBER OF RECORDS PER GROUP- 7
```

```
GROUP LENGTH IS- 512
```

```
SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT): E
```

```
END ISAM UTILITY.
```

10.2.5 Reorganizing a File (REORG)

As you add and delete records in an ISAM file, the sequential order and index are maintained, except, of course, there are no index entries for overflow records. When load exclusion, append, or overflow areas become filled, or performance deteriorates, you will want to restore these areas and place all records into indexed data groups. There are two ways to reorganize an ISAM file. You can rerun CREATE or run REORG. If you do not want to change the original specifications, REORG is the logical choice. REORG allows continued growth in the manner originally specified, and it does not permit you to change any of these specifications; however, if the resulting file is larger, you must specify how to allocate more space. Except for dialog, the process is the same as a CREATE.

10.2.5.1 Special REORG Characteristics

Input File

The input for a REORG is your present ISAM file. This file will be deleted, and the new file will be created.

Output File

The output file has the same name as the input file. There are temporary extensions assigned to the output files of .ROG and .RO1 through .RO7.

Note that the volumes containing data files must always be mounted for use on the same devices and units as those on which they were created (by CREATE or REORG). By using the RT-11 ASSIGN command for device names, you are not restricted to the same physical devices.

File Growth

The amount of growth is determined by a comparison of the number of records in the present file with the number of records in the file just after the last REORG or CREATE. Since the overflow area size does not change, the index expands or decreases by the space required for key entries. Since append and load exclusion are restored to their original values, the data area increases by a combination of records added from overflow, append, and load exclusion; or decreases as a result of records deleted.

Total File Requirements

Because the input file is deleted and because the output file has the same name as the input, the REORG function is similar to the CREATE function with deleted input and same input/output name. The files created are the same as the CREATE function under these conditions, namely:

filnam.CK1	(checkpoint file on system device)
filnam.CK2	(checkpoint file on system device)
filnam.TMP	(temporary file)
filnam.TM1	through .TM7
	(depending on the number of input data files)
filnam.ROG	(temporary REORG file)
filnam.RO1	through .RO7
	(depending on the number of input data files)
filnam.WK0	(work file)
filnam.WK1	(work file)

Until enough space is found to open all the required output files, the actual reorganization cannot start. After the output files are built, REORG enters the cleanup routine. The cleanup routine:

1. Writes a dummy checkpoint file.
2. Renames input files to .TMP and .TM1 through .TM7.
3. Renames output files to correct extensions.
4. Rewrites the output FCGs.
5. Deletes input files.
6. Checks to see if all output files are on the correct drives. (If yes, go to step 8.)
7. Moves files, if possible, to the correct device. After every successful move, it rewrites the FCG. If there is not enough room on a device, it terminates the REORG and tells you which device did not have enough room.
8. Rewrites a correct and complete FCG and gives you a status.

If REORG fails during this process, see Section 10.2.5.4.

10.2.5.2 REORG Process and Dialog — To run REORG, type R in response to the ISMUTL selection message. The program asks you for the name of the ISAM input file, and then proceeds with the reorganization. During the process, you may be asked to provide more work file space, or be notified of a problem during the cleanup procedure associated with the deletion of the input file. If any of these occur, see Section 10.2.5.4. The last response you will have to make is in regard to allocation of space for the added records in the indexed data files. The following is the text that will appear for a reorganization:

INPUT FILE (DEV:NAME.EXT):

This is your ISAM file to be reorganized.

APPROXIMATELY XX ADDITIONAL BLOCKS OF DISK SPACE REQUIRED
A= ADD EQUALLY TO EACH DATA FILE - -
B= ADD TO LAST DATA FILE
C= ADD EXTRA DATA FILE
PLEASE RESPOND WITH A,B OR C:

If you respond with C, the following additional question is asked:

ASSIGN DATA FILE DEVICE:

Respond with a device to accommodate the added file requirements.

10.2.5.3 REORG Example — The ISAM file illustrated in Sections 10.2.3.4 and 10.2.4.2 (CREATE and STATUS examples), is used to show the effect doing a reorganization has on a file with records in overflow, load exclusion, and the append area.

The reorganization dialog consists essentially of specifying the input file and of allocating additional storage space. The automatic status response shows the file after the reorganization.

The new file has the same 348 records as indicated in Section 10.2.4.2, but now the overflow area has been restored to a full 100 groups. Although not shown, the load exclusion record spaces and append area have also been restored. All the records are now in indexed data groups. Consequently the data file has increased by 22 groups and the index file by one group.

The REORG Example:

```
.R ISMUTL
```

```
CTS300 ISAM UTILITY PROGRAM, Vnn-nn
```

```
SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT): R  
INPUT FILE (DEV:NAME.EXT): XDATA1.ISM
```

```
APPROXIMATELY 23 ADDITIONAL BLOCKS OF DISK SPACE ARE REQUIRED
```

```
A= ADD EQUALLY TO EACH DATA FILE - - B= ADD TO LAST DATA FILE
```

```
C= ADD EXTRA DATA FILE
```

```
PLEASE RESPOND WITH A,B OR C:
```

```
10-MAY-83
```

```
KEY LENGTH IS- 8
```

```
RECORD LENGTH IS- 72
```

```
LEVELS OF INDEXING IS- 1
```

```
LOAD EXCLUSION FACTOR- 2
```

```
KEY STARTS AT LOCATION- 1
```

```
DUPLICATE KEYS ALLOWED- NONE
```

```
CURRENT NUMBER OF RECORDS- 348
```

```
GROUPS ALLOCATED TO OVERFLOW- 100
```

```
GROUPS REMAINING IN OVERFLOW- 100
```

```
NUMBER OF RECORDS PER GROUP- 7
```

```
GROUP LENGTH IS- 512
```

```
SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT): E
```

```
END ISAM UTILITY.
```

	A	
	FILE LOCATION	GROUPS ALLOCATED
	INDEX FILE DK	105
	DATA FILE -1 DK	100

10.2.5.4 Handling REORG Problems

Failure During Cleanup

If a machine malfunction or a forced termination occurs during REORG, it is still possible to complete the operation. Do not delete any temporary files, work files, or checkpoint files and just start the REORG again and specify the input file name. The program searches for a file with the CK1 or CK2 extension and, when found, restarts and completes the function.

Insufficient Space to Open Output Files

There are three ways you can deal with this situation. The method you use depends on your file and your system. The possible solutions are as follows.

1. Eliminate all unnecessary files, squeeze the disk(s), and retry ISMUTL. This process makes use of the checkpoint files.
2. Eliminate all the unnecessary files, including the input and checkpoint files, squeeze the disk(s), and start the entire process over. To do this, you must have a backup copy of the input.
3. Write a small program to convert your ISAM input file into a sequential file and, using this file, use CREATE to produce the desired ISAM file.

Insufficient Work File Space

REORG attempts to open output files on the same devices as the input file counterparts. If this is not possible, due to space limitations, a work device is requested via the message:

```
WORK FILES REQUIRE x BLOCKS OF DISK SPACE  
WORK FILE DEVICE:
```

Respond to this with a device name. If the specified device does not have enough space, the question will be repeated. Be sure that none of the devices made available to REORG have files with the same name as the input file; files with .ROG extension (or .RO1 through .RO7); or files with .CKn extensions. Any existing file of this same name will be deleted.

Until enough space is found to open all the required files (including the index build work files), reorganization cannot start.

10.2.6 Exiting from ISMUTL (EXIT)

When you no longer need ISMUTL, you may return to the monitor or the operating system by responding to the ISMUTL selection message with an E. The response is:

```
END ISAM UTILITY
```

In chain mode, the utility automatically chains to the specified user program.

10.2.7 Miscellaneous ISMUTL Capabilities

10.2.7.1 STATUS and REORG in Chain Mode — STATUS and REORG can be automated. If you end your program with ISMUTL as the optional argument with the STOP statement, control is automatically passed to the ISAM utility program. The following occurs when ISMUTL is chained to:

- If the chaining program was not detached, nothing is displayed on the terminal unless a fatal error occurs. The fact that a fatal error has occurred appears on the terminal, but the actual error is reported in a log file. This log file exists only when chaining to ISMUTL for a CREATE; it does not exist for a REORG or STATUS. The file is called XXX.LOG where XXX is the name of the ISAM output file.
- If the chaining program was detached, nothing is displayed on the terminal. A log file as above exists for a CREATE.
- See Section 10.2.7.2 for the special case where an auto-create is being performed.

Within your user program, you must have previously sent, with the SEND statement, a record of directions to ISMUTL. The record must contain specific field sizes in a certain order. Examples for REORG and for STATUS are shown below. They illustrate the required format for the record.

Example of a record sent to do a REORG:

NOTE

A REORG results in a file with the same append, overflow, and load exclusion choices as in the original CREATE. Therefore, the resulting file is usually larger. Allocation of space for this growth must be specified in field ADBLR in the passed record. Otherwise the REORG will fail.

USER PROGRAM (USPROG)

RECORD REORG

FNCT, A1,'R'	;DO A REORG
CHNFLG,D1,1	;0= DO NOT CHAIN TO USER PROGRAM
	;SET USRPG TO SPACES
	;1= CHAIN TO USRPG
FILIN, A17,'DK:ISAM.ISM '	;YOUR ISAM FILE
USRPG, A17	;PROGRAM TO CHAIN TO
ADBLR, A1	;ADDITIONAL SPACE (A, B, OR C)
EXPDV, A6	;DEV IF C (SPACES IF A OR B)
WORKNM,D1	;NUMBER OF WORK FILES
WORKDV,8A6	;(1-8 DEVICES) (3 CHAR DEVICE NAME,
	;NO COLON !)
	;IF THE NUMBER IS LESS THAN 8,
	;REMAINDER IS FILLED IN WITH SPACES
PROC	
.	
.	
.	
SEND (REORG,'ISMUTL',TERM)	;TERM = NUMBER OF TERMINAL
.	;YOU ARE CURRENTLY
.	;RUNNING ON
.	
STOP 'ISMUTL'	;STOP THIS PROGRAM START
	;UP ISMUTL

Example of a record sent to do a STATUS:

In the following example, the first time the user program (USPROG.SAV) is run there is no message to receive, so the record STATUS is therefore sent to ISMUTL with instructions to chain (see fields CHNFLG and USRPG) to USPROG.SAV.

At the end of the ISMUTL STATUS function, a record is sent to the user program (specified by USRPG), and ISMUTL chains to USPROG. The record STAT in the example must contain the fields shown in order to receive the STATUS information sent by ISMUTL.

```

USER PROGRAM (USPROG)
  RECORD STATUS
  FNCT, A1, 'S'           ;DO A STATUS
  CHNFLG, D1, 1          ;0= DO NOT CHAIN TO USRPG (SET
                        ;USRPG TO SPACES)
                        ;1= CHAIN TO USRPG
  FILIN, A17, 'DK:ISAM.ISM ' ;YOUR ISAM FILE
  USRPG, A17, 'USPROG '   ;FILE TO CHAIN TO
  , A56                  ;FILLER
  APDFLG, D1             ;0= DO NOT RETURN APPEND RECORDS,
                        ;NON ZERO VALUE
                        ;= RETURN - APPEND RECORDS

  RECORD STAT
  CUROFL, D6             ;NO. OF UNUSED OVRFLOW GROUPS
  TOTOFL, D6             ;NO. OF GROUPS ALLOCATED TO OVRFLOW
  TOTREC, D12            ;CURRENT NO. OF RECORDS IN FILE
  ORGREC, D12            ;ORIG NO OF RECORDS IN FILE
  APDREC, D8             ;REMAINING NO. OF RECORD SPACES IN
                        ;THE APPEND AREA

  TERM, D2

  PROC
  .
  .
  .

  RECV(STAT, LABEL)     ;CHECK FOR STATUS INFO. IF NONE,
                        ;SEND MESSAGE

  .
  .
  .                       ;OTHERWISE PROCESS AND OUTPUT
                        ;STATUS

  STOP                  ;STOP WITHOUT CHAIN

  .
  .
  .

  LABEL, XCALL TNMBR(TERM)
  SEND(STATUS, 'ISMUTL', TERM) ;TERM = TERMINAL YOU ARE ON

  .
  .
  .

  STOP 'ISMUTL'         ;STOP THIS PROGRAM START UP ISMUTL

```

The STATUS function will perform considerably faster if the remaining number of record spaces in the append area is not requested. When APDFLG in record STATUS contains zero, the remaining number of record spaces will not be returned.

10.2.7.2 Auto-CREATE — The CREATE function requires considerable operator interaction to produce a file. The auto-create capability eliminates most, or all, of this interaction. There are many applications for this capability. One use would be the situation where the parameters of a file are known, and similar files are to be built. It could also be used when some predictable small changes are to be made to an ISAM file, and it would be convenient to avoid numerous sessions with the dialog. Another situation would be a “turnkey” system, in which the operator is not permitted to do a manual create. A final example of an application would be an auto-create to do a reorganization with different parameters than originally specified; because as a file matures the distribution of records to be added often changes.

The auto-create is selected with an answer of YES to the first question in the CREATE dialog. This is followed by a request for an auto-filename. The auto-filename is a data file constructed by the user which contains a list of answers for the expected ISAM CREATE questions. Upon specifying the auto-filename, the CREATE process continues to completion, or until an invalid answer is encountered.

In either single-user or time-shared operation, access to the auto-filename or response file can be done manually with the YES answer to the first CREATE question. Access may also be made automatic (programmed). Programmed access to the auto-filename is a function of whether you are operating under single-user or time-sharing.

Under SUD the program selection (R ISMUTL) and initial responses (C, YES, auto-filename) are supplied from an indirect file. The responses to the CREATE dialog are then supplied by the auto-filename. Below is an illustration of this indirect file:

```

!INDIRECT FILE ISMCRT
R ISMUTL !RUN ISMUTL
C        !CREATE FUNCTION
Y        !AUTO OPTION
filnam   !AUTO FILENAME (CREATE QUESTION RESPONSES)

```

Time-shared operation does not support indirect file operation, but a message can be sent to ISMUTL to supply the initial responses in the same manner as with REORG and STATUS when they are run in chain mode. The YES answer to the auto-create question is assumed by ISMUTL when this mode is chosen. The record sent must contain the specific field sizes in the order shown below:

```

                USER PROGRAM (USPROG)

                RECORD CREATE
FNCT, A1,'C'           ;DO A CREATE
CHNFLG,D1             ;0 = DO NOT CHAIN TO USRPG (SET
                     ;USRPG TO SPACES)
                     ;1 = CHAIN TO USRPG
FILIN, A17            ;AUTO-FILENAME (RESPONSE FILE)
USRPG, A17            ;CHAIN FILE

PROC
.
.
.
SEND (CREATE,'ISMUTL',TERM) ;TERM = NUMBER OF TERMINAL
                           ;YOU ARE CURRENTLY
                           ;RUNNING ON
STOP 'ISMUTL'           ;STOP THIS PROGRAM START
                           ;UP ISMUTL

```


The following is an example of an auto-file. In the previous examples, this is filnam in the indirect file for SUD, or field FILIN in the message sent from the time-shared program. This auto-file creates the same ISAM file as shown in the example in Section 10.2.3.4. The responses in the file must be exactly the same as they would be in a manual create. The only exception is the specification of DK: for the default device rather than a carriage return. This is done to make reading the file easier.

```
.R DKED
*AFILE.CTL=           ;AFILE is filnam or FILIN
XDATA1.DDF           ;input file
N                   ;do not delete input file
XDATA1.ISM           ;output file
250                 ;number of input records
7                   ;number of records per group
2                   ;load exclusion

Y                   ;fill out group
200                 ;records in append area
100                 ;overflow groups
8                   ;key length
1                   ;key location
N                   ;no duplicate keys
DK                  ;allocate index file to default dev
DK                  ;allocate data file to default dev
80                  ;allocate 80 blocks for the data file
E                   ;exit ISMUTL
<GOLD/COMMAND>
EXIT
```

If an ISMUTL auto-create is called via chaining, the normal dialog which appears on the screen will be written to the log file (in this case: XDATA1.LOG). This file will show all display input and output, and will help reveal problems with execution of ISMUTL in the auto-create mode.

If any answer to the response file is invalid, the CREATE will fail. In the manual mode, an invalid answer will result in the question being repeated; in auto mode, the program terminates. If the file is being created in the detached mode (time-shared) when an error occurs, the program chain will be broken. A detailed study of the program queries and appropriate responses must be made.

Table 10-1 summarizes the methods you can use to create an ISAM file.

Table 10-1
ISMUTL CREATE Modes

	System	
Mode	SUD	TSD
Auto- mated	ISMCRT.COM where ISMCRT.COM contains: R ISMUTL C Y auto-filename	In the calling program: RECORD CREATE FNCT, A1, 'C' CHNFLG, D1 FILIN, A17, 'auto-filename' USRPG, A17 PROC SEND(CREATE, 'ISMUTL.TSD') STOP 'ISMUTL'
Semi; Auto- mated		R ISMUTL C Y auto-filename
Manual		R ISMUTL C N Respond to the CREATE dialog

CHAPTER 11

SORTGEN

11.1 INTRODUCTION

At one time or another, you will probably be faced with a file whose records are not in the order you need for a particular use. Or you may find that you have two or more files containing information in identically structured records that you want to combine into one file to form a common data base. A sort program facilitates the reordering of a file, and a merge program is used to combine files having the same record structure.

CTS-300 includes tools that enable the user to develop a sort or merge program. One sort method is described in this chapter. Another (the SORT utility) is described in Chapter 12. SORT is particularly useful for users with an XMTSD system.

With SORTGEN the development of a sort or merge program starts with a user-written control file that describes the parameters of the operation. The SORTG program, with the control file as input, generates the variable data division of the desired sort or merge program. This file and SORTM.DBL, which is the procedure division of the sort or merge program, are then compiled to produce an object file. This object file is linked like any other such file to produce the usable sort or merge program. When the program is run, the specified files are sorted or merged, as requested.

11.1.1 Characteristics

Several features of SORTGEN should be pointed out:

- Files may be sorted by multiple keys, and the sort for each key may be in ascending or descending order.
- An optional key sort (TAGS sort) is available which allows a sort of key fields only, without transporting entire records across work files.
- Optional commands are provided to allow the user to define such operating parameters as memory allocation, work files, and so forth.
- The SORTGEN sort or merge program developed by the CTS-300 user has the optional ability to receive a control record sent from another program. This control record facilitates changes in selected sort or merge parameters at run-time.

When you use SORTGEN, you must be aware of some limitations:

- SORTGEN operates only on fixed-length records.
- The caret symbol (^), which is 136 octal, is the internal control character used in the sort for the end-of-string marker. If used as the first character of a record, sort results will be unpredictable.

- The sort program is not able to sort a multivolume sequential file, because sort requires a CTRL/Z at the end of all input files.
- The sort or merge program is not able to sort or merge records with packed keys. Compressed records can be handled as long as the record lengths and displaced key(s) are correctly described, and the sort keys remain unpacked.
- The output of a sort or merge is a sequential file. Sorting an ISAM file will not produce another ISAM file as output.

11.1.2 Chapter Organization

The remainder of this chapter is structured in the general sequence required to develop a sort or merge program. It is comprised of two sections. Section 11.2, the Control File, covers control file characteristics and requirements. The individual commands within the control file are explained, and examples are given where necessary. The commands are then summarized. This is followed by Section 11.3, SORTGEN Program Development and Use, which covers the development procedure. Development consists of using the routines supplied by CTS-300, SORTG and SORTM, to integrate the control file into the desired program. The last steps of the procedure are basically routine compilation and linking. Examples are used to illustrate both features and correct usage. Last of all, the use of a control record for chain mode operation is presented.

11.1.3 Error Messages

Error messages associated with SORTGEN are documented in the *CTS-300 System Message Manual*.

11.2 THE CONTROL FILE

The sort or merge that you want operates on information supplied by you, the user. The control file describes the input file(s), output file, sort key(s), record format, and other necessary parameters and options. This control file is created by you with the aid of an editor.

The control file is composed of five required control commands and six optional control commands. A different control file must be built for each sort or merge. There are situations where another control file may involve just a simple change of input file name, or it may require a change of most or of all the parameters. Whenever any command is changed, the control file must go through the procedure of being processed by SORTG, compiled along with SORTM, and linked to produce the usable program.

Each control command in the control file, with the exception of RECORD, must be contained on one line. With regard to the record command, RECORD: must be on one line and each of the field definitions must be on a single, separate line. A control file cannot contain more than 50 lines.

The characteristics and use of all eleven commands are discussed on an individual basis in the following subsections. Control command options are shown in brackets. Comment lines are allowed on any command line if preceded by a semicolon. Examples are given for each command requiring clarification.

Your first sort or merge exercise will be easier if you avoid the optional commands. The five required commands are summarized below:

INPUT

This command is used to specify the input file(s) for the sort or merge (multiple input files imply a merge operation).

OUTPUT

This command is used to specify the name of the sorted or merged file that results from running the sort or merge program.

RECORD

The program must know the record structure of the input file(s) it is dealing with and the location of the key(s) within the record. This command specifies these parameters.

KEYS

This command defines the order in which the file is to be sorted, in relation to the chosen key(s).

END

This command identifies the end of the control file.

The required commands are presented in alphabetical order in the following sections. There is no required order for the commands (except that INPUT must be the first command and END must be the last command).

DETACH (Optional)

11.2.1 DETACH (Optional)

The DETACH command is an optional command that causes the sort or merge program to detach from its associated terminal. This allows you to execute another job while a sort or merge is running.

The format is:

DE[tach]:

There are no arguments.

The characteristics of this command are:

- DETACH is ignored under SUD operation.
- Any detached program that produces terminal output will hang until it is attached.
- If error messages are generated, they will not be displayed until the program is attached.
- DETACH occurs only once. If you reattach, your terminal is dedicated to the attached program until it terminates. See Chapter 5 of this manual for information on the ATTACH command.

END

11.2.2 END

The END command indicates the end of the control file. The format is:

EN[d]:

There are no arguments. The only restriction on the command is that it must be the last one in the control file.

EXECUTE (Optional)

11.2.3 EXECUTE (Optional)

EXECUTE is an optional command that specifies the name of a program to be executed (chained to) after the sort or merge program terminates.

The format is:

EX[ecute]:filespec

where:

filespec specifies the name of the program to be executed.

- If EXECUTE is omitted, control returns to the system monitor or the run-time system.
- EXECUTE may be overridden. See Section 11.3.6.

Example:

EXECUTE:MMENU.SAV

Indicates that program MMENU.SAV on the default device is to be executed upon completion of the sort or merge operation.

INPUT

11.2.4 INPUT

The INPUT command is the first command in a control file. The file to be sorted or the files to be merged are specified here. The specification of a single file indicates that a sort is desired. The specification of two or more files indicates that a merge is to be performed.

The format is:

```
IN[put]:filespec1[,filespec2...,filespec7][(count)]
```

where:

- | | |
|-----------|---|
| filespec | is the first input file specification. If this is the only file specified, the operation is a sort of this file. |
| filespec2 | through filespec7 is one or more additional file specification(s) up to a maximum of seven total files or 70 characters. If included in the command, the operation is a merge of the files specified here. Files specified in addition to the first file are separated by commas. |

NOTE

Files to be merged must be sorted prior to merging.

- | | |
|---------|--|
| (count) | is optional when used with sequential files. It specifies the number of records, starting with the first record, to be sorted or merged. If multiple files are specified, the count argument must either be specified for all of the files or for none of the files. The number must be enclosed by parentheses. |
|---------|--|

NOTE

The count argument must be used for an ISAM input file(s) and must, in this case, contain only the letters ISM.

- Mixed file types cannot be used as input to a merge operation. That is, ISAM and sequential files cannot both be specified in the same merge operation.
- The count argument can be overridden at run time by a program that chains to your sort or merge program. See Section 11.3.6.

Example:

INPUT:RK0:PAY.DDF(2400)

Sorts the first 2400 records of file RK0:PAY.DDF.

INPUT:LABR.DDF

Sorts file LABR.DDF on the default device. The entire file is sorted.

INPUT:RK1:LABM.DDF(3000),RK1:LABT.DDF(4000)

Merges the first 3000 records of file RK1:LABM.DDF and the first 4000 records of file RK1:LABT.DDF.

INPUT:RK0:PAY.ISM(ISM)

Sorts the ISAM file on RK0 called PAY.ISM.

KEYS

11.2.5 KEYS

The KEYS command specifies which fields within the input file records are to be used to control the order of the output file. These fields are known as control keys.

The format is:

```
KE[ys]:fldnam1[-],[fldnam2[-]...,fldnam8[-]]
```

where:

fldnam1 through fldnam8

indicates the field(s) to be used as the control field(s) (or control key(s)).

- Each fldnam must be a name of a field defined in the RECORD command.
- The first fldnam specified becomes the major control key; the last fldnam specified becomes the minor control key; and the fldnams between become intermediate control keys in the order specified.
- No more than eight control keys may be specified.
- The space character sorting sequence depends on whether it is an alpha or a decimal field. An alpha space is an octal 40 and a numeric zero is an octal 60. If a decimal field is used, a blank and a zero will be interpreted the same way.

- is an optional argument associated with a given fldnam that, if included, causes records to be ordered in descending sequence by that fldnam. If not included, the records are ordered in ascending sequence.

Example:

```
KEYS:NAME,DATE-
```

Specifies a sort or merge using the field called NAME as the major control field, ordered in ascending order, and the field called DATE as the minor control field, ordered in descending order. Names appearing first in the alphabet would appear first in the file, and for each name category, the records would be arranged with the most recent date appearing first.

OUTPUT

11.2.6 OUTPUT

The OUTPUT command specifies the name you want to give the file that is the output of the sort or merge operation.

The form is:

OU[tput]:filespec

where:

filespec is the name of the sequential output file.

- The output file can have the same name as the input file only if the input file is sequential and the input file is being sorted. The input file is overwritten.
- The output file cannot have the same name as the input file if a merge is performed; if a TAGS:LIST, or TAGS:INDEX sort is being done (see Section 11.2.10); or if the input file is an ISAM file. A SORTG error message will result if any of these is attempted.

In the case of a TAGS:SORT, the error message will appear at run time and only if there is not sufficient free space for a temporary copy of the input file.

- The output file is always a sequential file, even if the input files are ISAM files.

Example:

OUTPUT:RK1:WEEKLY.DDF

The sorted or merged output is written as a file named WEEKLY.DDF on device RK1.

RECORD

11.2.7 RECORD

The RECORD command defines the format of the records in the input file(s). The format for each file is identical to the DIBOL record format, except that no initial value may be specified.

The format is:

RE[cord]:

field def 1

.

.

.

field def n

where:

field def 1 to field def n

are field definitions for each key field to be used in the subsequent sort or merge; for fields to be used as filler to position the key fields within the record; and for fields to fill out the record to the total length of the input record.

- Each field definition is on a separate line.
- Each field definition is formed in accordance with the rules for a DIBOL data division statement.
- The entire record must be described.
- Unnamed fields can be used as space fillers to place named fields in the proper position within the record.

Example:

RECORD:

FILLA,	A5	;fill characters
NAME,	A20	
,	A35	;fill characters
BADGE,	A5	

This record command defines a record of 65 characters. Positions 6-25 contain a key field named NAME, and positions 61-65 contain a key field named BADGE. The other positions in the record may contain any number of other fields, but they are of no interest to the sort or merge program.

SPACE (Optional)

11.2.8 SPACE (Optional)

The SPACE command is an optional command that controls the amount of additional memory available for a sort or merge operation.

The format is:

SP[ace]:kbytes

where:

kbytes is a decimal number used to specify, in thousand byte segments, the total amount of main memory to be available to a sort or merge program. This includes the DIBOL program and all buffers, but does not include the DIBOL interpreter, monitor, or any other requirement.

- The default memory size, if the SPACE command is not used, is 16 K bytes.
- If the default of 16 K bytes is used, and the record is too large to allow a minimum of three records to be held in the work buffers, SORTG suggests, via a message, the amount of space required to support the number of work files requested. You then have the option of reducing the number of work files or of increasing the space.

However, if you have used the SPACE command, and a minimum of three records cannot be in the work buffers, SORTG overrides the SPACE command and automatically optimizes the internal work areas without notifying you.

- The sort or merge program is designed to work within artificial limits imposed by TSD. If the sort requirements exceed the memory space available, a message (as above) will appear, suggesting new values for space and work file allocation.
- For faster operation, add 1 K byte of memory for every work file assigned in addition to the default of three work files.
- Files with very large records may require more than the default of three work files (see Section 11.2.11). Specify 1 K byte of additional memory for each added file.
- For TSD operation, the SPACE option should be used with consideration for the memory requirements of other users.

Example:

SPACE:18

Indicates that a sort or merge program may use up to 18 K bytes of memory.

SU (Optional)

11.2.9 SU (Optional)

SU is an optional command that specifies that the sort or merge program is going to be run as a Single-User DIBOL program. Single-user operation is faster than time-shared operation, for several reasons, and should be used for large files.

The format is:

SU:

There are no arguments.

The characteristics of SU are:

- Compared to time-sharing, SU operation is faster, because more memory can be used. There is no contention for memory or disk resources.
- I/O buffer allocation is automatically doubled with the SU option.
- The SU option forces six work files. More may be specified with the WORK command, but it is not advised for sorts that utilize random access I/O (TAGS:SORT).
- The SU option overrides the SPACE command and automatically optimizes the internal working areas.
- If you use the SU option with the RT-11 XM monitor, you may receive a message indicating there is not enough memory. If this happens, use the RT-11 SJ monitor.

TAGS (Optional)

11.2.10 TAGS (Optional)

The TAGS command is an optional command that allows a sort to be performed using only the keys instead of the entire record. A key sort offers substantial advantages in execution speed and minimized disk requirements, since the entire file need not be manipulated. The TAGS option should be used for most sorts.

The format is:

TA[gs]:type

where:

type is either SORT, LIST, or INDEX.

TAGS:SORT

specifies a sort that produces a sorted version of the input file. The key or keys are first sorted, and then a pass is made to write each complete record in the new output file.

- TAGS:SORT must not be used on an ISAM file.
- A TAGS:SORT requires much smaller work files than a normal sort.
- A TAGS:SORT can provide a significant performance advantage over a normal sort when working with a record of approximately 80 characters and a short key. With larger records, the advantage is greater.
- The final pass to write the output file consumes much of the time required for a TAGS:SORT. This pass is eliminated with either the TAGS:LIST or the TAGS:INDEX sort below.
- The output file name must be different from the input file name.

TAGS:LIST

specifies a sort that produces an output file which consists of a seven-byte record number. This file can then be used to access the original file. The TAGS:LIST is basically like the TAGS:SORT but without the pass required for the post-sort write.

- Each of the records in the output file contains a record number identifying the record in the original file that corresponds to the relative position of this number in the output file. That is, the third record in the output file contains the record number for the record in the original file that has become third in the desired sort sequence.
- Leading zeros in the output file are encoded as spaces.
- The output file name must be different from the input file name.

TAGS:INDEX

specifies an output file consisting of the key field(s) specified in the KEYS command and a corresponding seven-byte relative record number, as in the TAGS:LIST command. The record format consists of the major key, the intermediate key(s), the minor key, and the relative record number.

- If your original file is an ISAM file, use the TAGS:INDEX option and add the ISAM key as a minor sort key. It has little effect on sort speed, and you can then access the original ISAM data file by means of the sorted TAGS:INDEX output file. That is, you can select a record on the basis of any chosen key in the TAGS:INDEX output file, and then access the record in the ISAM file using the corresponding ISAM key.
- The output file name must be different from the input file name.

WORK (Optional)

11.2.11 WORK (Optional)

The WORK command is an optional command that specifies both the number of work files to be used for a sort and the devices on which the work files are to reside.

The format is:

```
WO[rk]:[number][,dev1:...,devn:]
```

where:

number is a decimal number between three and seven, inclusively, that specifies the number of work files to be allocated. If the number (or the WORK command itself) is omitted, the default is three work files on the default device.

dev1: through *devn*:

specifies a list of one or more devices to which the work files are allocated. Dev: must be a valid device name and must include the colon. If dev: is omitted, all work files are allocated to device DK:.

- The first work file is assigned to the first device in the list. The second work file is assigned to the second device, and so forth. The same device may be specified more than once.
- If the number of devices specified is less than the number of work files, devices are allocated to files by starting over with the first device specified. File assignments cycle through the device list, until all the files have been assigned to devices.
- If the number of devices assigned exceeds the number of work files, the excess devices are ignored.
- If a merge is specified, the WORK command is ignored.
- With a file of approximately 5000 to 10,000 records, four to six work files, rather than the default of three, will result in a measurable performance improvement.

Example:

```
WORK:4
```

Allocates four work files, all on device DK:.

```
WORK:3,RK2:
```

Allocates three work files, all on device RK2.

```
WORK:3,RK1:;RK2:;RK3:;RK4:
```

Allocates three work files, one each on devices RK1, RK2, and RK3. RK4 is ignored, since only three work files are specified.

WORK:3,RK1:,RK2:

Allocates three work files, the first on device RK1, the second on device RK2, and the third on device RK1.

11.2.12 Summary of Control File Commands

Below is a summary (in one possible order for logical use) of the control file commands with optional commands indicated.

Command	Optional	Comments
IN[put]:filespec1[...filespec7] [(count)]	No	First command
OU[put]:filespec	No	Name of the file resulting from the sort or merge operation
RE[cord]: field def	No	Fields must be in specified order
KE[ys]:fldnam1[-] [...fldnam8[-]]	No	Maximum of 8
DE[tach]:	Yes	Ignored under SUD operation
EX[ecute]:filespec	Yes	
SP[ace]:kbytes	Yes	Ignored with SU option
SU:	Yes	SU overrides SPACE allocation
TA[gs]:SORT :LIST :INDEX	Yes	Fast sort
WO[rk]:[number] [,dev1:...,devn:]	Yes	Ignored for merge
EN[d]:	No	Last command in control file

11.3 SORTGEN PROGRAM DEVELOPMENT AND USE

Now that you have developed a control file, it must be processed by SORTG. The output of SORTG is compiled together with SORTM, and the result linked to produce your sort or merge program. The program is then run to perform the sort or merge. The details of these steps are contained in the following sections.

11.3.1 SORTG

SORTG is a CTS-300 supplied file (SORTG.SAV) that generates the data division of your sort or merge program, using your control file as input. It is an executable program that asks you for the name of its input file and of its output file. The dialog sequence follows:

```
.R SORTG
SORT GENERATOR VAnn-nn
INFILE =
```

Respond with the name given to your control file when you created it with the editor. There is no default extension. The response will be:

```
OUTFILE =
```

Respond with the desired file specification of your sort or merge program (SMPROG.DBL for the sake of the following discussion).

After you provide the output file name, SORTG generates the output file unless an error is detected. Errors for SORTG are listed in the *CTS-300 System Message Manual*. If an error is detected, a message will be displayed indicating that output has been suspended. This message will be followed by the error message. The error message is usually preceded by the line number of the control command in error.

11.3.2 Compiling with SORTM

SORTM.DBL is a CTS-300 supplied file and is the procedure division of your sort or merge program. SORTM and the output resulting from running SORTG (SMPROG.DBL) must both be compiled to produce an OBJ file. That is:

```
.R DICOMP
*SMPROG = SMPROG,SORTM/O
```

whose output is SMPROG.OBJ.

11.3.3 Linking

The .OBJ file produced by the compiler must be linked to run in either single-user or time-sharing mode.

for single-user:

```
.LINK SMPROG,ODIBOL,DIBOL
```

whose output is: SMPROG.SAV, or for time-sharing:

```
.LINK/EXE:SMPROG.TSD SMPROG,ODIBOL,TDIBOL/BOT:100000
```

whose output is: SMPROG.TSD

You will want to run REDUCE on this TSD file to delete unused blocks.

11.3.4 Running the Sort or Merge Program

The sort or merge program is run like any other SUD or TSD/XMTSD program:

```
.R SMPROG or .RU dev:SMPROG
```

The input file(s) specified in the control file is sorted (or merged) to produce a file with the specified output name. Errors generated during execution are those listed for SORTM in the CTS-300 System Message Manual.

11.3.5 Example

The following is an example illustrating all the steps required to generate and run a sort program.

There are five steps:

1. Create the control file:

```
.R DKED
*FILE.CTL=
IN:DATA1.DDF           ;the unsorted input data file
OU:SDATA1.DDF         ;the desired sorted file
RE:                   ;record definition follows
,      A5              ;space filler
NAME, A20             ;a field to be used as a sort key
,      A35             ;space filler
BADGE,A5              ;a field to be used as a sort key
SU:                   ;the sort program will be
                     ;run in a single-user environment
KE:NAME,BADGE         ;major control key followed by
                     ;a minor control key
EN:                   ;end of control file
<GOLD/COMMAND>
EXIT
```

2. Run SORTG

```
.R SORTG
SORT GENERATOR VAn-nn
INFILE = FILE.CTL
OUTFILE = SORT.DBL
```

3. Compile:

```
.R DICOMP  
*SORT = SORT,SORTM/O
```

whose output is: SORT.OBJ

4. Link:

```
.LINK SORT,ODIBOL,DIBOL
```

whose output is: SORT.SAV

5. Run the sort program:

```
.R SORT.SAV
```

When you run SORT.SAV file, DATA1.DDF will be sorted as specified into a new file SDATA1.DDF.

11.3.6 SORTGEN in Chain Mode

Certain parameters in the SORTGEN Control File can be changed at run time. This feature is particularly useful in the single-user environment where chaining and message route-through are desired. The parameters that can be changed are:

- You can change the record count (optional) that was originally specified in the control file INPUT command and name a program to be executed after the sort or merge is completed. The named program overrides the one that was originally specified in the control file EXECUTE command.
- You can select the EXECUTE command, as originally specified, as the only command to be executed. No sort or merge takes place.
- You can specify a message to be sent to the program specified in the EXECUTE command of the SORTGEN Control File. The message is passed to the program at the completion of the sort or merge.
- You can change the record count, name a program to be executed following the sort or merge, and specify a message to be sent to that program.

The operations listed above are accomplished via a control record that is sent to the sort or merge program prior to its execution. The operation performed is determined by the value of the first character in this record (the OPCODE field).

To change the specified record count and the program to be executed after the sort or merge:

The control record form is:

```
RECORD SCR  
OPCODE,A1,'%'  
COUNT, D6  
NEXT, A14
```

where:

- OPCODE is the character “%”.
- COUNT is a decimal value that is inserted as the control file INPUT command record count. Any previous value is overwritten. If COUNT is set to zero, the value originally specified in the control file INPUT command is used.
- NEXT is the file specification for the file to replace the program originally specified by the control file EXECUTE command.

To specify the EXECUTE command only:

The control record form is:

```
RECORD SCR
OPCODE,A1,'@'
```

where:

- OPCODE is the character “@”.

The original control file EXECUTE command is the only command executed. No sort or merge takes place.

To send a message:

The control record form is:

```
RECORD SCR
OPCODE,A1
,      A20
,      AXXX,'message'
```

where:

- OPCODE is a character other than a %, a @, a #, or a space.
- XXX indicates the field (message) size. The size of the message is limited to 100 characters.
- message is the message.

The sending program chains to the sort or merge and the specified sort or merge is performed. The message is then routed to the program originally specified in the EXECUTE command (if one was specified) and the sort or merge chains to this specified program.

To change the record count, name the program to be executed, and to send a message:

The control record form is:

```

RECORD SCR
OPCODE,A1,'#'
COUNT, D6
NEXT, A14
, AXXX,'message'
    
```

where:

OPCODE is the character "#".

COUNT is a decimal value that is inserted as the control file INPUT command record count. Any previous value is overwritten. If COUNT is set to zero, the value originally specified in the control file INPUT command is used.

NEXT is the file specification for the file to replace the program originally specified by the control file EXECUTE command.

XXX indicates the field (message) size. The size of the message is limited to 100 characters.

message is the message.

Summary of parameter change via SORTGEN control record:

FUNCTION	CHARACTER				
	@	%	#	other character	blank
Normal SORT					X
EXECUTE only	X				
Change program to be executed and record count		X			
Send a message				X	
Send a message, change record count, and change program to be executed			X		

CHAPTER 12

SORT

12.1 INTRODUCTION

SORT is a program written in MACRO whose purpose is to provide a fast easy-to-use sort or merge capability. SORT has several advantages over the SORTG SORTM pair of programs (the sort/merge utility provided prior to Version 7) documented as SORTGEN in Chapter 11.

12.1.1 Characteristics

SORT features:

- Operates as a stand-alone program under RT-11 (SORT.SAV).
- Operates in the XMTSD environment (SORT.TSD).
- Can use a SORTG control file as input (see SORTGEN, Chapter 11).
- Does not require a generation phase.
- Accepts sort parameter specification via terminal input in command line form.
- Accepts sort parameter specification under DIBOL application control from either a file or a passed record.
- Can be run as a background program.
- Does not require an end-of-file marker.
- Chooses the optimum sort environment for best performance.

SORT limitations:

- Cannot be run under TSD.
- Does not sort files with packed decimal keys.
- Does not sort multivolume sequential files.
- Does not necessarily retain the original order of records that have duplicate sort keys.

SORT requirements:

- Excluding working storage, SORT.SAV requires approximately 9 KW.
- Excluding working storage, SORT.TSD requires approximately 6 KW.
- In addition to the SORT program itself, a minimum of 2 KW of memory is required for working storage.

12.1.2 Chapter Organization

The remainder of this chapter is comprised of three sections. The first, Section 12.2, Using SORT, discusses the general nature of the specification command line, the second, Section 12.3, SORT Commands, discusses sort/merge parameter specification, and the last section, Section 12.4, Examples, illustrates the use of SORT.

12.1.3 Error Messages

Error messages associated with SORT are documented in the *CTS-300 System Message Manual*.

When you are running SORT in an interactive mode all errors will appear on your terminal. However, when you are running SORT from either a control file, a message string, or as a detached program, the routing of your error messages depends on the SORT IDENT command.

If the IDENT command is not used, the errors will be sent to the terminal (if the sort is running detached, the errors will appear when you next attach).

If the IDENT command is used and an error occurs in the SORT command decoder, SORT error 16 will be sent to the program identified by IDENT and the sort will be stopped at that point. If errors occur after the sort has begun, the error messages will be sent to the program. The errors will be identified by a three-digit number. The only SORT errors that can occur under these conditions are error numbers 16, 29, 31, and 32. They will appear as DIBOL negative numbers (01v = 16, 02y = 29, 03q = 31, and 03r = 32) to distinguish them from DIBOL errors which can also occur. DIBOL errors are reported with their appropriate number.

12.2 USING SORT

With SORT there are two methods of specifying the parameters for the sort or merge you want:

- First, a character string using delimiters to separate the parameters can be used. This will be referred to in this chapter as the SORT method or format.
- Second, the sort parameters can be specified in a format identical to that used with the SORTGEN software (SORTG control file). This second method will be referred to in this chapter as the SORTG method or format.

Either of the two methods can be entered interactively at the keyboard, from a control file specified at the keyboard, or via a message sent from another program using the DIBOL SEND statement. For SORT the message sent can be the name of a record that contains SORT commands.

12.2.1 Keyboard Level Input

SORT is executed as either a stand alone program (RT-11 monitor level) or as an XMTSD program with the following commands:

for a stand alone system (RT-11):

```
.R SORT
```

or, for an XMTSD system:

```
*SORT
```

The response in either case is the SORT prompt:

```
SORT >
```

At this point the sort or merge operation parameters can be specified. This specification could take one of following forms:

- Sort parameters specified in character string format with delimiters (SORT method).
- Sort parameters specified in the SORTG control file format (SORTG method). This specification requires that a special SORT input identifier (DIBOL) be the first entry to distinguish this input from that of the SORT method.
- The name of a file containing a character string with a list of sort commands separated by delimiters (the SORT method format).
- The name of a file containing parameters in the format required by SORTG. This requires that the special SORT input identifier (DIBOL) be included with the file specified or as the first entry in the file. This is to distinguish this input from that of the SORT method.

Alternatively, the parameters, in the four possible formats just described, can be entered prior to the carriage return in a keyboard command level mode with no SORT prompt:

for a stand alone system:

```
.R SORT parameters
```

or, for an XMTSD system:

```
*SORT parameters
```

12.2.2 Application Program Level Input

It is possible, using the DIBOL SEND statement, to send to SORT (from an application program) a record, control file name, or message (character string) that specifies the desired sort parameters.

```
SEND (recnam,'SORT',n)
```

or

```
SEND ('@filespec.ext','SORT',n)
```

where:

recnam is the name of the record containing the SORT commands. This record contains an input character string with delimiters (SORT method).

- The record is limited to a maximum size of 350 characters.
- A record cannot be used to transmit parameters in the SORTG format.

@filespec.ext

is a control file.

- A file in the SORT format contains a SORT input character string with delimiters.
- If the special input identifier (DIBOL) is used with either the file specification or in the file itself, the control file can be a file in the SORTG format.
- There is no default extension, one must be specified.

n

is a numeric character that either determines that the SORT program is to be started detached or determines the terminal to which it is to be attached. See the SEND statement in the *DIBOL-83 Language Reference Manual* and Chapter 5 in this manual.

12.2.3 Parameter Specification

Whether entered at the keyboard, as a control file, or as the contents of a record, the option selections, file identifiers, and other specifiers of the sort or merge are collectively identified in SORT as “commands”. SORT commands are discussed in detail in the next section.

The following illustrates, in general terms, the way these commands are used when specifying sort/merge parameters discussed in the previous two subsections. The only command specifically mentioned is the “DIBOL” command because its use is somewhat different from the others.

Input from a keyboard:

1. Input entered interactively (with SORT prompt) in the SORT format:

```
*SORT  
SORT> command/command/...
```

.

.

.

```
SORT> end command
```

or (without the SORT prompt):

```
*SORT command/command/...
```

.

.

```
SORT> command/.../end command
```

2. Input entered interactively (with the SORT prompt) in the SORTG format:

```
*SORT
SORT> DIBOL
SORT> SORTG parameter
.
.
.
SORT> SORTG end parameter
```

or (without the SORT prompt):

```
*SORT DIBOL
SORT> SORTG parameter
.
.
.
SORT> SORTG end parameter
```

3. Input identifying a control file in the SORT format:

```
*SORT
SORT> @filespec
```

or (without the prompt):

```
*SORT @filespec
```

4. Input identifying a control file in the SORTG format:

a. If the file does not contain DIBOL as the first line:

```
*SORT
SORT> @filespec/DIBOL
```

or (without the prompt):

```
*SORT @filespec/DIBOL
```

b. If the file contains DIBOL as the first line:

```
*SORT
SORT> @filespec
```

or (without the prompt):

```
*SORT @filespec
```

Input from an application program:

1. Input of a control file that is in the SORT format:

SEND('@filespec','SORT',TTn)

2. Input of a record that is in the SORT format):

SEND(recnam,'SORT',TTn)

3. Input of a control file (including DIBOL as the first line in the file) that is in the SORTG format:

SEND('@filespec','SORT',TTn)

or (if the control file does not contain DIBOL):

SEND(@filespec/DIBOL,'SORT',TTn)

12.2.4 Stopping SORT

SORT can be stopped from the keyboard only when the sort parameters were specified from the keyboard in the interactive mode and the LOCKCC command was not specified. A double CTRL/C under these conditions will abort the sort or merge. When SORT receives a message either from the keyboard (SORT prompt not used) or from an application program, it cannot be stopped regardless of whether the LOCKCC command was used or not.

12.3 SORT COMMANDS

The SORT commands are listed below.

Command	Meaning
ASSIGN	Establishes the location (device(s)) of the intermediate work files.
CHAIN	Specifies a program to be executed automatically upon termination of SORT.
COUNT	Identifies the number of records to be processed starting with the first record in the file.
DETACH	Requests that SORT be run detached.
DIBOL	Identifies the sort specifications as being in the SORTG/SORTM format. (If used, must appear first).
END	Identifies the end of the SORT command entries. (Must appear last).
IDENT	Identifies the name of a program to receive a message when SORT terminates. Used with TTNUM and MESSAGE.
INPUT	Identifies the input file(s).

ISAM	Identifies the input file as an ISAM file.
KEYS	Defines the size, location, data type, and sort order.
LOCKCC	Disables the ability to stop SORT from the keyboard with a double CTRL/C when operating in the interactive mode.
MESSAGE	Specifies the text of a message to be sent by SORT to the program specified by IDENT command. Used with IDENT and TTNUM.
OUTPUT	Identifies the file to contain the result of the sort or merge operation.
PAD	Defines the character used as a filler in an over-allocated file.
RECLEN	Defines the length of the record in characters.
SPACE	Defines the size of memory to be allocated for SORT use (the SORT default choice is the maximum available).
TAGS	Specifies an operation in which keys and associated record numbers are used instead of entire records.
TEMP	Specifies a file used for temporary storage when a file is being sorted into itself.
TTNUM	With XMTSD systems specifies the number of the terminal attached to the program. Used with IDENT and MESSAGE.
VERSION	Identifies the current version of the SORT program.
WORK	Specifies the number of intermediate work files to be used in place of the value chosen by SORT. Used only when the PAD command is specified.

The following are the general rules which govern the use of SORT commands:

- Some commands are required; others are optional. These are identified in following subsections.
- Only the first two characters are required.
- The order of entry of commands is not important except for DIBOL (if used) and END which must be the first and last, respectively.
- Line length in the command line format at the keyboard cannot exceed 80 characters. If additional specification is needed, upon typing <CR> SORT will prompt for additional input. SORT will continue to prompt in this manner until the END command is specified.
- Commands are separated by a "/" or a <CR> . A "/" will be used throughout this chapter.
- Commands that identify filespecs use an "=" as a separator.
- Multiple occurrences of filespecs use a "," as separators.
- Commands that specify values use a ":" as a separator.
- Comments can be inserted using a ";" or an "!".
- Multiple occurrences of commands on the same line are separated by commas. This applies to INPUT, ASSIGN, and KEYS.

Section 12.3.22 summarizes all of the above plus the individual command descriptions that follow.

ASSIGN (Optional)

12.3.1 ASSIGN (Optional)

ASSIGN is used to specify the location of intermediate work files.

The format is:

```
AS[sign]:dev[,dev,...]
```

where:

dev[,dev,...]

is the physical or logical device name(s) of the device(s) on which the work files are to be assigned.

- Up to eight device names can be specified. If more than one device is specified, the specifications are separated by commas.
- If the ASSIGN command is not used, the work files are placed on DK:.
- The device name can be any logical or physical device name from one to three characters long (the colon is optional) such as A:, BB:, or CCC:.
- The first work file is assigned to the first device specified. The second work file is assigned to the second, and so on. The same device can be specified more than once.
- If the number of devices specified is less than the number of work files, devices are allocated to files starting again with the first device specified. File assignments cycle through the device list until all the files have been assigned to devices.
- If the number of devices exceeds the number of work files, the excess devices are ignored.
- If a merge is specified, the ASSIGN command is ignored.

Advantages of using the ASSIGN command:

- A device (possibly allocated solely to work file storage) can be used rather than DK: which may have little room left.
- The work files can be allocated to place most of the work files on one device and the rest on another by using a format such as the following:

```
ASSIGN:DL1:,DL1:,DL2:
```

which places approximately (depending on the exact number of work files) twice as many of the files on DL1: as on DL2:.

CHAIN (Optional)

12.3.2 CHAIN (Optional)

CHAIN is used to specify the name of a program to be automatically run after SORT is complete.

The format is:

CH[ain] = filespec

where:

filespec is the file specification of the program to be automatically executed upon termination of the sort.

- If CHAIN is not used, the system returns to the XMTSD prompt or the RT-11 monitor.
- If SORT is running detached (XMTSD), the program started via CHAIN will also run detached.

COUNT (Optional)

12.3.3 COUNT (Optional)

COUNT is used to provide SORT with the number of records to be sorted. If COUNT is not used, SORT bases its record count, and, therefore, disk space needs, on the block count of the file. SORT obtains the block count from the operating system and calculates the number of records from this value.

The format is:

CO[unt]:value

where:

value is a decimal value that represents the total number of records to be sorted.

- Counting of records starts with the first record of the input file.
- COUNT is meaningless when used with an ISAM input file and is ignored if specified.
- If the PAD command is used with the COUNT command, the COUNT value is forced to zero and SORT obtains the record count from the input file block size.

NOTE

The COUNT command must be used with caution when sorting a file into itself. This is because a CTRL/Z is written at the end of the file as determined by the count value and this may not be the end of valid data.

DETACH (Optional)

12.3.4 DETACH (Optional)

DETACH causes the sort to proceed as a detached program.

The format is:

DE[tach]

There are no arguments.

The characteristics of this command are:

- DETACH is ignored for stand-alone operation.
- If the IDENT command is not used, any error message will appear when you next attach.

DIBOL (Optional)

12.3.5 DIBOL (Optional)

DIBOL indicates that the sort parameters in a sort specification file or entered interactively are in the SORTG control file format.

The format is:

DI[bol]

There are no arguments.

The characteristics of this command are:

- The DIBOL command must be the first entry if interactive input is used.
- The DIBOL command must either follow the filespec identifying the control file or be the first entry in the control file itself if a control file is used in SORTG control file format.

END

12.3.6 END

END is a required command that terminates the command interpreter phase of the sort. The sort or merge operation proceeds following receipt of the END command.

The format is:

EN[d]

There are no arguments.

Characteristics of this command are:

- The END command must be the last one specified.

IDENT (Optional)

12.3.7 IDENT (Optional)

IDENT is used in conjunction with the TTNUM and MESSAGE commands. It identifies the name of a program to receive a message from SORT during a detached sort or when SORT terminates.

The format is:

ID[ent]= filespec

where:

filespec is the name of the program to which the message is sent.

INPUT

12.3.8 INPUT

INPUT is a required command that identifies the source file or files.

The format is:

```
IN[put] = filespec1[,...,filespec6]
```

You can also use the form:

```
IN[put] = filespec1
```

```
.
```

```
.
```

```
.
```

```
IN[put] = filespec6
```

where:

`filespec` is the file specification of a source file.

- File specifications are separated by commas if the first form is used.
- Up to six files can be specified.
- If an input file of length zero is specified, an error is generated.
- If an empty ISAM file is specified as an input file, no error is generated.

If more than one file is specified it identifies a MERGE operation. The additional characteristics of a merge are:

- ISAM files cannot be merged.
- Files to be merged must first be sorted using the same keys as those to be used in the merge.
- The ASSIGN and TAG commands are meaningless and, therefore, ignored.

ISAM (Optional)

12.3.9 ISAM (Optional)

ISAM is used to identify the input file as an ISAM file.

The format is:

IS[am]

There are no arguments.

The characteristics of this command are:

- An ISAM file can be TAGS:INDEX sorted but not TAGS:SORT or TAGS:LIST sorted. A TAGS:INDEX sort produces a file consisting of the specified keys and a relative record number. The relative record number is not used in this case, but a TAGS:INDEX sort using the ISAM key as the minor key allows the access of ISAM records on the basis of this key.
- An ISAM file cannot be sorted into itself.
- ISAM files cannot be merged.
- A sort of an ISAM file always produces a sequential file.
- The PAD and COUNT commands are not meaningful if specified for an ISAM file. If specified they will be ignored.

KEYS

12.3.10 KEYS

KEYS is a required command that identifies the fields of the record that are to be used as the basis of the sort.

The format is:

KE[ys]:ABS.L[,ABS.L,...]

where:

ABS.L[,ABS.L,...]

is the key designator(s). If multiple keys are to be used the designators are separated by commas in the command string.

- The first key identified becomes the major key. Following keys are minor keys sorted in the order in which they are specified.
- Up to eight keys can be specified.
- The format for each key designator is:

ABS.L

where:

- A is a single character data type identifier and can take the values: A for alphabetic or D for decimal.
- B is a single character identifying sort direction: A for ascending or D for descending.
- S is a decimal value identifying the starting character position of the key within the record. The value of S can be from 1 to the size of the record.
- L is a decimal value defining the length of the key. It can have a value of 1 to (record size-1)-S.

The separator between the S and L in the key designator is a decimal point.

LOCKCC (Optional)

12.3.11 LOCKCC (Optional)

LOCKCC prevents you from stopping a sort whose parameters have been specified in the interactive mode. Normally such a sort can be stopped by entering two CTRL/C's at the keyboard.

NOTE

A sort initiated via message from an application program (or from the keyboard as a string without using the SORT prompt) cannot be stopped from the keyboard.

The format is:

LO[ckcc]

There are no arguments.

MESSAGE (Optional)

12.3.12 MESSAGE (Optional)

MESSAGE is used in conjunction with the IDENT and TNUM commands and specifies the text of a message that will be sent by SORT to the program specified by IDENT upon termination of the sort.

The format is:

ME[ssage]:string

where:

string is the text of the message.

- The total size of the message is limited to 40 characters.
- The message cannot contain a "/" character.
- The first three characters of the message are reserved for SORT. If the sort or merge completed without error, the first three characters are set to zero. If the sort or merge produces an error, the first three characters are set to the SORT or DIBOL error number (see Section 12.1.3).
- If the IDENT command is missing, the message is ignored.
- If TNUM is missing, the message is sent to the program specified in IDENT.
- If the MESSAGE command is missing, but the IDENT command is present, SORT will generate the 3-character message of zero if the sort were successful or the DIBOL error number if unsuccessful.

OUTPUT

12.3.13 OUTPUT

OUTPUT is a required command that identifies the name of the output file.

The format is:

OUT[put] = filespec

where:

filespec is the file specification of the destination file.

It has the following characteristics:

- The output file specification must be different than the input file specification unless you intend to sort the file into itself. Sorting into the input file has the advantage of minimizing disk fragmentation and retaining record space that may have been allocated for file expansion.

NOTE

The COUNT command must be used with caution when sorting a file into itself. This is because a CTRL/Z is written at the end of the file as determined by the count value and this may not be the end of valid data.

NOTE

When a TAGS:SORT is performed on an XMTSD system with input and output files having the same name and a temporary file specified (the only way a TAGS:SORT can be performed), ensure that the input file is terminated with a CTRL/Z. Otherwise you will be prompted by a request to mount the next volume for the temporary file.

- If a TAGS:LIST or TAGS:INDEX is used, the output file must be different than the input file - that is, a sort so specified cannot be sorted into itself.
- If a TAGS:SORT is performed on a file being sorted into itself, the TEMP command must also be used.
- You must specify an output file when sorting an ISAM file because you cannot sort an ISAM file into itself.
- The output file will be a sequential file even if the input file was an ISAM file.
- SORT appends a CTRL/Z to the output file except for files sorted into themselves which are sorted using the PAD command.

PAD (Optional)

12.3.14 PAD (Optional)

PAD is used to define a user-inserted end-of-file marker instead of the CTRL/Z that DIBOL files use. It allows identification by an application program of records in files that have been overallocated with padded record slots. These records will not be included as part of the sort.

The format is:

PA[d]:spclchar

where:

spclchar is the user-defined filler character used to mark the end of the file (the beginning of the remaining overallocated area).

- The pad character must be the first character after the last valid data record.
- If no pad character is defined, the default character that SORT looks for is the usual CTRL/Z.
- If no end-of-file character (pad character or CTRL/Z) is found, SORT processes to the end of the last block unless it finds a null character as the first character in the next logical record slot.
- If the PAD command is used, SORT does not optimize the number of work files. In this case three work files are assigned. The WORK command can be used to specify another value.

RECLLEN

12.3.15 RECLLEN

RECLLEN is a required command that declares the length of each record in the input file(s).

The form is:

RE[clen]:value

where:

value is a decimal number whose minimum value is 1 and whose maximum value is determined by the DIBOL record size limit and the amount of available memory.

NOTE

For sequential files, the value given for RECLLEN must include any record terminator(s) (that is, for sequential / random files, a carriage return / line feed).

Records in an ISAM file do not contain a terminator. The size of an ISAM record can be obtained using the STATUS option in ISMUTL.

SPACE (Optional)

12.3.16 SPACE (Optional)

SPACE is used to direct SORT to use less than the maximum amount of memory available at run time. This memory is for the SORT program and SORT internal work areas.

The format is:

SP[ace]:value

where:

- value is a decimal value that represents the maximum size of memory to be used by the SORT program and memory work units in terms of 1024 (1K) word increments.
- If the maximum specified exceeds the available space, the memory allocated is automatically reduced.
 - The value for stand-alone systems can be in the range of 12 to a maximum of 28 KW.
 - The value for XMTSD systems can be in the range of 8 to 16 KW.
 - The value used must be no smaller than the program size plus 2.
 - When SORT is run as a background program, it normally runs slower because of space limitations and the fact that it is a lower priority program.

TAGS (Optional)

12.3.17 TAGS (Optional)

TAGS is used to establish the type of tag sort to be performed. The size of the intermediate work files is greatly reduced when a TAGS sort is done.

The format is:

```
TA[gs]: SORT
        LIST
        INDEX
```

where:

- | | |
|-------|--|
| SORT | produces a sorted copy of the input file using only the key fields, a seven-digit relative record numbers, and a <CR><LF> terminator as entries in the work records. Although this method results in a full record sort, it is much slower than an ordinary sort. Its advantage is in using smaller intermediate work files. That is, it saves file space. |
| LIST | produces a file of records consisting of relative record numbers in specified sequence. This is a fast sort. |
| INDEX | produces a file consisting of records containing the sort keys, a seven-digit relative record number, and a <CR><LF> terminator. This can be used to access the original record. This is a fast sort. |

Characteristics:

- If a merge is specified, the TAGS command is ignored.
- TAGS:SORT must not be used on an ISAM file.
- TAGS:LIST sort and TAGS:INDEX sort require separate input and output files.
- IF a TAGS:SORT has input and output files with the same name, the TEMP command must be used.
- If the input file is ISAM. specify the ISAM key as a minor sort key and use the TAGS:INDEX sort. Any sort key can then be used as the basis for record selection and the corresponding ISAM key is available for record retrieval.

12.3.18 TEMP

TEMP is used to specify a file to be used for temporary storage when a sort operation has identical input and output files; that is, when a file is being sorted into itself.

The format is:

TE[mp] = filespec

where:

filespec is the specification of the temporary work file.

- No qualifiers may be used with the file specification.
- If TEMP is not used, and an error occurs during the sort operation, you may end up with no input file and a corrupted output file.
- If TEMP is used, the temporary storage file becomes the output file when the sorting operation is complete.
- TEMP must be used with a TAGS:SORT.

NOTE

When a TAGS:SORT is performed on an XMTSD system with input and output files having the same name and a temporary file specified (the only way a TAGS:SORT can be performed), ensure that the input file is terminated with a CTRL/Z. Otherwise you will be prompted by a request to mount the next volume for the temporary file.

- If the input file and output file are different, TEMP is ignored.

TTNUM (Optional XMTSD Systems)

12.3.19 TTNUM (Optional XMTSD Systems)

TTNUM is used in XMTSD systems in conjunction with the IDENT and MESSAGE commands. It specifies the number of the terminal attached to the program specified in the IDENT command. A message will be sent to this terminal upon termination of the sort.

The format is:

TT[num]:n

where:

n is the number of the terminal.

- If the number is not specified, the message is simply sent to the program with the name specified by IDENT.
- If the IDENT command is missing, TTNUM is ignored.
- TTNUM is ignored in stand-alone operation.

VERSION (Optional)

12.3.20 VERSION (Optional)

VERSION is used in an interactive mode to display the current version number and patch level of the SORT program.

The format is:

VE[rsion]

There are no arguments.

The display output format is:

CTS300 SORT Vnn-nn

WORK (Optional)

12.3.21 WORK (Optional)

WORK is used to establish a value for the number of work units SORT will use. This value overrides the optimum number chosen by SORT. The WORK command is used when the PAD qualifier is also specified. This is because use of the PAD qualifier is the only case where the sort is not optimized.

The format is:

WO[*rk*]:value

where:

value is a decimal number in the range of 3 to 8.

- When the PAD command is used three work files are chosen by SORT.
- For a full record sort, the size of each work file is at least as large as the input file (or as large as a file whose size is based on the record count if the COUNT command is used) and possibly as much as one and a half times the size of the input file. TAG sorts of any kind will produce smaller work files because they are allocated on the basis of the secondary record length.

12.3.22 Summary

Response for command line input:

command/command/.../END

Response for a file containing command line formatted input (SORT method):

@filespec.ext

Response for a file containing control file formatted input (SORTG method):

@filespec.ext/DIBOL

SORT Command Summary			
Command	Required	Special Characteristics	Meaning
ASSIGN: dev[,dev,...]	N	overrides default of DK: assigned in sequence	Establishes the location (device(s)) of the intermediate work files.
CHAIN = filespec	N		Specifies a program to be executed automatically upon the termination of SORT.
COUNT:value	N	overrides SORT value	Identifies the number of value records to be processed for partial file sorting.
DETACH	N	ignored under stand alone operation	Requests that SORT run detached.
DIBOL	N	must be in the first line	Identifies the file specified as being in the SORTG/SORTM format.
END	Y	must be the last command	Identifies the end of the SORT command entries.

IDENT = filespec	N		Used with TTNUM and MESSAGE to specify the program to receive the specified message upon termination of SORT.
INPUT = filespec[,filespec,...]	Y		Identifies the input file(s).
ISAM	N		Identifies the input file as an ISAM file.
KEYS:ABS.L [,ABS.L,...]	Y	<p>A = data type Alpha Decimal</p> <p>B = direction Ascending Decending</p> <p>S = start position</p> <p>L = key length</p>	Defines the data type, size, location, and sort direction.
LOCKCC	N	interactive mode operation only	Prevents termination of a sort or merge with a double CTRL/C.
MESSAGE:'string'	N		Used with TTNUM and IDENT to specify the message to be sent upon termination of SORT.
OUTPUT = filespec	Y	always a sequential file	Identifies the file to contain the result of the sort or merge.

PAD:spclchar	N	replaces default of CRTL/Z; cannot be used with ISAM	Defines the character used as a record identifier in an over-allocated file. Overrides COUNT.
RECLEN:value	Y		Defines the length of the record in characters (includes the terminator for sequential files).
SPACE:value	N	overrides SORT value range XMTSD: 8-16 KW range stand alone: 12-28 KW	Defines the size of memory to be allocated for SORT use (in place of the default chosen by SORT).
TAGS: SORT LIST INDEX	N	SORT - sorted copy LIST - record numbers INDEX - keys plus relative rec. num.	Specifies an operation in which keys are used instead of entire records.
TEMP	N		Specifies a storage file for use when sorting a file into itself.
TTNUM:n	N		Used with IDENT and MESSAGE to specify the terminal to which the specified message is to be sent upon termination of SORT.
VERSION	N		Displays SORT version and patch level.
WORK:value	N	overrides SORT value value 3-8	Specifies the number of intermediate work files to be used in place of the default value chosen by SORT. Used in conjunction with PAD.

12.4 EXAMPLES

First, examples will be shown illustrating the methods that could be used to specify a sort using the same sort parameters for each.

All the examples shown using keyboard input are for an XMTSD system using the SORT prompt.

The information given:

Input file

FILE1.DDF

Number of records to sort

1200

Record size

98 characters (including carriage return and line feed characters)

Key identification

Field name, ID

Size, 12 alpha characters

Starting position, 1st character in the record

Relative importance, major key

Field name, DATE

Size, 6 decimal characters

Starting position, 13th character in the record

Relative importance, minor key

Nature of sort (ascending/descending)

Ascending by ID

Descending by DATE

Output file

FILOUT.DDF

Example 1 Command line input:

```
*SORT <CR>
```

```
SORT> IN = FILE1.DDF/CO:1200/RE:98/KE:AA1.12,DD13.6 <CR>
```

```
SORT> OU = FILOUT.DDF/EN
```

Example 2 Filespec input:

```
*SORT <CR>  
SORT> @SORTA1.SRT
```

where SORTA1.SRT contains:

```
INPUT = FILE1.DDF <CR>  
COUNT:1200 <CR>  
RECLEN:98 <CR>  
KEYS:AA1.12,DD13.6 <CR>  
OUT = FILOUT.DDF <CR>  
END
```

Example 3 Filespec input using a SORTG control file:

```
*SORT <CR>  
SORT> @OLDSRT.SRT/DIBOL
```

where OLDSRT.SRT contains:

```
IN:FILE1.DDF(1200)  
OU:FILOUT.DDF  
RE:  
ID, A12  
DATE, D6  
FILL, A78  
KE:ID,DATE-  
EN:
```

or (if DIBOL is the first line of the file):

```
SORT> @OLDSRT.SRT
```

where OLDSRT.SRT contains:

```
DIBOL  
IN:FILE1.DDF(1200)  
OU:FILOUT.DDF  
RE:  
ID, A12  
DATE, D6  
FILL, A78  
KE:ID,DATE-  
EN:
```

Example 4 Via a record (message or character string) sent from another program:

This is probably the most common method of sort parameter specification. In this example the SEND statement in the application program starts SORT as a detached program.

```
SEND (SRT,'SORT',-2)
```

where record SRT is:

	RECORD SRT
INPUT,	A13,'IN = FILE1.DDF/'
COUNT,	A8,'CO:1200/'
LEN,	A6,'RE:98/'
KEYS,	A17,'KE:AA1.12,DD13.6/'
OUTPUT,	A14,'OU = FILOUT.DDF/'
END,	A2,'EN'

or

INPUT,	A12,'IN = FILE1.DDF'
COUNT,	A8,'/CO:1200'
LEN,	A6,'/RE:98'
KEYS,	A17,'/KE:AA1.12,DD13.6'
OUTPUT,	A14,'/OU = FILOUT.DDF'
END,	A3,'/EN'

NOTE

As seen above, the "/" separator can be placed prior to (except the first) or following each command in the message or character string.

CHAPTER 13

STATUS

13.1 INTRODUCTION

STATUS is a utility used to obtain message information for all systems and, for TSD and XMTSD, information about the run-time system.

13.1.1 Features

The information obtained using STATUS is determined by the option you choose. The specific kinds of information available are:

All systems (SUD, TSD, XMTSD):

- List of STATUS options.
- Pending messages.
- Information on a specific message.
- Removal of a message.
- Exit from STATUS.

TSD and XMTSD systems only:

- Available free memory and its relative location.
- Total active jobs.
- Information on a specified active job.
- Characteristics of the current version of a time-shared run-time system.
- Termination of an active job.
- On a time-shared system STATUS runs only if there is an attached terminal.
- Information is automatically updated for certain options, and the frequency of update is user selectable.
- For time-shared systems STATUS outputs to either a terminal or a line printer.

STATUS requires 10 K bytes of memory.

13.1.2 Chapter Organization

The remainder of this chapter is comprised of three sections. Section 13.2, GENERAL, presents the basic start-up procedures, errors, and options. Section 13.3, Time-Shared Systems, presents options available for TSD and XMTSD systems and explains how to use the STATUS utility to gather information on your time-shared system. Section 13.4, Single-User Systems, presents the options available for SUD systems.

13.2 GENERAL

13.2.1 Starting and Running STATUS

13.2.1.1 Time-shared Systems — Type the following:

```
*STATUS
```

The response is:

```
OUTPUT TO LINE PRINTER? ENTER Y/N:
```

Respond with a Y or an N. The default is N and indicates that you want the output device to be the terminal. A Y response indicates that you want the output device to be a line printer in addition to the terminal. Line printer selection produces a further question:

```
ENTER PRINTER NUMBER. (1-4):
```

Enter the number for the desired printer on your system. The printer must have been specified during SYSGEN. If the printer were not specified in SYSGEN, or if the printer is busy, an appropriate message is displayed. If an invalid printer number (not 1-4) is entered, the system prompts for the number again.

After selection of output device, a list of options is displayed or printed, as with Option H, see Section 13.4.1.

13.2.1.2 Single-user Systems — Type the following:

```
.R STATUS
```

After selection of output device, a list of options is displayed or printed, as with Option H, see Section 13.4.1.

13.2.2 Errors

Errors are generated by STATUS for the following:

- invalid options
- invalid number identifier for a job or message
- illegal responses

For further information see the *CTS-300 System Message Manual*.

13.2.3 Options

STATUS functions as a result of the options you select. A list of options is initially displayed every time STATUS is run, and a brief option list is displayed at the bottom of the screen each time STATUS displays information. Only one option can be selected at a time. STATUS recognizes a maximum of three characters to designate an option.

To select an option enter the letter(s) (and number, if appropriate) of the option desired followed by <CR> .

The individual options in the following sections are shown with typical output for illustration.

13.3 TIME-SHARED SYSTEMS (TSD AND XMTSD)

13.3.1 Option D

Option D displays a line requesting you to choose the frequency with which the STATUS display is updated.

The response is a display at the top of the screen:

ENTER UPDATE FREQUENCY IN SECONDS(1-99):

There is no default with this option. However, the value for D when the system is initialized is 5 seconds. Your response determines how often new information is displayed for the STATUS options which are updated. These are options F, J, and M.

13.3.2 Option F

Option F displays the amount of free memory and identifies its location.

Typical Option F response:

```
FREE MEMORY                               DD-MMM-YY HH:MM:SS
HIGH MEMORY:    145408 BYTES
LOW MEMORY:     0 BYTES
STATUS.TSD:     10240 BYTES IN HIGH MEMORY
```

ENTER OPTION (F,H,J,K,M,T,X OR D):

The amount of core memory available in extended memory (HIGH) and in lower memory (LOW) is shown. The high memory value will always be zero unless you have an XMTSD system. The display illustrated is for an XMTSD system running in the foreground. The size and location of STATUS indicates additional memory that will be available when STATUS is terminated.

13.3.3 Option H

Option H presents a list of the STATUS options.

The response is:

CTS300 SYSTEM STATUS PROGRAM Vnxx-xx

OPTIONS: (F) FREE MEMORY
(H) HELP
(J) ACTIVE JOB LIST
(JX) ACTIVE JOB NO. DESCRIPTION
(KX) KILL AN ACTIVE JOB
(M) MESSAGES PENDING
(MX) CONTENTS OF A MESSAGE
(MA) CONTENTS OF ALL MESSAGES
(MKX) KILL A MESSAGE
(T) TSD PARAMETERS
(X) EXIT
(D) SET UPDATE FREQUENCY FOR STATUS

ENTER OPTION (F,H,J,K,M,T,X OR D):

13.3.4 Option J

Option J displays the complete list of active jobs on the system.

Typical Option J response:

JOB NO.	JOBNAME	TERM.NO.	SIZE	MEM.LOC.	DD-MMM-YY	HH:MM:SS
1	KBDLDR	0	2048	HIGH	STATUS	
2	DK :STATUS.TSD	1	10240	HIGH	KBWAIT	
3	KBDLDR	2	2048	HIGH	ACTIVE	
4	DK :PTRTST	DET	10240	LOW	KBWAIT	
5	DK :DKED .TSD	3	26624	LOW	LPWAIT	
6	DK :IOTEST	DET	2048	HIGH	KBWAIT	
7	DK :IOWAIT	DET	2048	LOW	HIBER	
	<MESSAGE AREA>		2048	LOW	IOWAIT	
					ACTIVE	

ENTER OPTION (F,H,J,K,M,T,X OR D):

JOB NO. is a number assigned by STATUS to identify a program. JOBNAME is displayed in usual file specification format (dev:filnam.ext). If there is no job running at a terminal, and if the terminal is able to receive a job request, that terminal will show a pseudo job indicated by the name KBDLDR. The size of the job is in bytes and, in XMTSD only, includes I/O buffers. Memory location is either high or low. The entry in the status column indicates one of five states for the program (see below). The last line in the example shows the message buffer allocation for XMTSD systems only. If there are no messages pending, the message area report does not appear. In this case it shows there is 2 K bytes of buffer space used for this purpose.

Option J status information:

The status column indicates one of following five possible states for the job:

STATE	MEANING
ACTIVE	The job is running.
IOWAIT	The job is waiting for completion of an I/O event.
KBWAIT	The job is waiting for keyboard input.
LPWAIT	The line printer job is waiting for completion. If this status continues it could indicate operator intervention required (paper out, off line, etc.)
HIBER	The job is attempting to output to a terminal. <ul style="list-style-type: none">• If the job is detached it will remain in this state until a terminal is attached.• If the job is attached to a terminal, output is being attempted at a rate the terminal cannot handle. This is a temporary condition and could be caused by a high volume of output or by operator selection of NO SCROLL.
ERRORH	The job is attempting to access the run-time system error handling code. If STATUS is continuously in this state, notify your DIGITAL representative.

13.3.5 Option Jx

Option Jx displays information on a specific job. The STATUS job number is specified by a numeric value for x and can be one or two characters. Use the J option to determine the STATUS job number.

Typical Option Jx response:

JOB NO. 5
JOBNAME DK :DKED .TSD
SIZE: 26624
STATUS: KBWAIT

DD-MMM-YY HH:MM:SS
TERM.NO.: 3
LOCATION: HIGH

CHANNEL	DEV:FILNAM.EXT	FILES FLSIZE	OPEN MODE	TOT.USERS	UPD.USERS	BLOCK
4	DK :TEST .DDF	161	I	1	0	8
11	DK :TEST .BAK	283	O	1	0	1
7	DK :TE .S10	275	O	1	0	1

ENTER OPTION (F,H,J,K,M,T,X OR D):

This message repeats the job number (5, selected by specifying option J5) on the first line. The second line specifies the job file specification and the terminal number (if detached, it is indicated by DET). The size of the job in bytes is on the third line followed by the location (high or low memory). The fourth line is job status (ACTIVE, IOWAIT, KBWAIT, LPWAIT, HIBERNATE, or ERRORH).

A list of the files open for this job is displayed under the FILES OPEN heading. The information includes: the channels open for the job, the device and files being accessed, the file sizes, the mode of access, the total users of that file, the number of update users of that file, and the number of the block currently in use by the program. This is a real-time display updated with a frequency determined by Option D.

If the job number specified is valid but no files are open, a message appears indicating that no files are open. If an invalid job number is specified, a message will appear indicating that no job by that number exists.

13.3.6 Option Kx

The option Kx cancels (kills) the job indicated by the specified numeric value of x which can be one or two characters. This option permits you to terminate any job, regardless of whether or not the job is attached or detached, or on a master or a slave terminal. Use the J option to determine the number of the job you want to terminate.

When the job specified is running attached, the time-shared system version number is displayed once the job is terminated. When the job specified is running detached, there is no message.

Use the J or Jx option to verify the deletion.

NOTE

The Kx option operates like a double CTRL/C. Files opened in output mode are lost and files that were opened in update mode may not reflect the latest record updates.

13.3.7 Option M

Option M displays the contents of the message queue.

Typical Option M response:

NO. MESSAGE FOR PROGRAM	TERM.NO.	SIZE	DD-MMM-YY	HH:MM:SS
1 DK :TST .TSD	NONE	24		
2 DK :TST2 .TSD	NONE	88		
3 DK :TST3 .TSD	NONE	24		

ENTER OPTION (F,H,J,K,M,T,X OR D):

Each pending message is indicated on a separate line. Information provided includes message number (assigned by STATUS and used with the Mx option), destination program name, the number of the terminal for which the message is queued (if any), and size (in bytes).

If more than 22 messages are in the queue, the first 22 will appear and, after a short pause, the rest of the messages will be scrolled onto the screen. The display can be stopped using the SCROLL/NO SCROLL key (or CTRL/S and CTRL/Q).

If there are no messages for any program, a message is displayed indicating there are none.

13.3.8 Option MA

Option MA displays the contents of all messages in the queue.

Typical Option MA response:

NO. MESSAGE FOR PROGRAM	TERM.NO.	SIZE	DD-MMM-YY	HH:MM:SS
1 DK :TST .TSD	NONE	24		
THIS IS A MESSAGE				
2 DK :TST2 .TSD	NONE	88		
THIS IS MESSAGE TWO				
3 DK :TST3 .TSD	NONE	24		
THIS IS MESSAGE THREE				

ENTER OPTION (F,H,J,K,M,T,X OR D):

The first 80 characters (one line) of the message is displayed. If more than 11 messages are in the queue, the first 11 will appear and, after a short pause, the rest of the messages will be scrolled onto the screen. The display can be stopped using the SCROLL/NO SCROLL key (or CTRL/S and CTRL/Q).

13.3.9 Option Mx

Option Mx displays the contents of a selected message in the same format as Option M. The message is specified by a numeric value for x and can be one or two characters. Use the M or MA option to determine the message number.

Typical Option Mx response:

NO. MESSAGE FOR PROGRAM	TERM.NO.	SIZE
3 DK :TST3 .TSD	NONE	24
THIS IS MESSAGE FOUR		

ENTER OPTION (F,H,J,K,M,T,X OR D):

If the message number is not valid (that is, shown with the M option), a message is displayed indicating the number is invalid.

13.3.10 Option MKx

Option MKx deletes (kills) a message selected by the specified value of x which can be one or two characters. The message number is obtained by using Option M or MA. After deletion of the message, the Option M display is presented showing the remaining contents of the message queue.

13.3.11 Option T

Option T displays the operating parameters of the current version of the TSD or XMTSD run-time system.

Typical Option T response:

TSD PARAMETERS			DD-MMM-YY HH:MM:SS
MAX. NO. OF JOBS:	16	NO. OF TERMINALS:	10
MAX. NO. OF FILES:	32	MAX. NO. OF MESSAGES:	25
MAX. NO. OF DEVICES:	1	MAX FILES FOR UPDATE:	16
SLICE:	64	ISAM:	YES
SWAP USR:	NO	DDT:	YES
FORCED JOB STARTUP:	YES	IMPLICIT JOB START:	YES
CIS:	NO	MONITOR(SJ/FB/XM):	XM
FATAL JOB:	FATAL.TSD		

ENTER OPTION (F,H,J,K,M,T,X OR D):

The values displayed are the limits for the current version of the TSD or XMTSD system. Most of these reflect decisions made during CTSGEN and can be changed only with another CTSGEN; the exceptions are SLICE, CIS (Commercial Instruction Set), and USR. USR swapping is selectable in CTSGEN for TSD (XMTSD does not allow swapping, USR is locked in memory at all times).

In an XMTSD system, the entry for the maximum of files for update will be the same as the response to that question in CTSGEN. An XMTSD system requires that memory be preallocated for tables associated with opening files. The entry will be zero for a TSD system because the memory for these tables is allocated at run time.

The fatal job named is the special program selected during CTSGEN that runs in place of the application if a fatal error is encountered. If no program has been selected, this entry is blank.

13.3.12 Option X

This is the exit option. It allows control to return to the run-time system. When option X is selected, the TSD or XMTSD run-time system regains control of the terminal and displays the current version number and the asterisk prompt. Any program linked for time-shared operation can then be run.

13.4 SINGLE-USER SYSTEMS (SUD)

STATUS of single-user systems allows access to information related to messages. The information displayed is similar to that available for messages on time-shared systems. See Chapter 5, Section 5.3.2, for more information on how messages are handled in a single-user system.

13.4.1 Option H

Option H presents a list of the STATUS options.

The response is:

CTS300 SYSTEM STATUS PROGRAM Vnxx-xx

OPTIONS: (H) HELP
(M) MESSAGES PENDING
(MX) CONTENTS OF A MESSAGE
(MA) CONTENTS OF ALL MESSAGES
(MKX) KILL A MESSAGE
(X) EXIT

ENTER OPTION (H,M,MA,MX,MKX,X):

13.4.2 Option M

Option M displays the contents of the message queue.

Typical Option M response:

NO.	MESSAGE FOR PROGRAM	DATE	SIZE
1	TST .SAV	30-MAR-83	24
2	TST2 .SAV	30-MAR-83	88
3	TST3 .SAV	30-MAR-83	24

ENTER OPTION (H,M,MA,MX,MKX,X):

Each pending message is indicated on a separate line. Information provided includes message number (assigned by STATUS and used with the Mx option), destination program name, the date the message was queued, and the message size (in bytes).

If more than 22 messages are in the queue, the first 22 will appear and, after a short pause, the rest of the messages will be scrolled onto the screen. The display can be stopped using the SCROLL/NO SCROLL key (or CTRL/S and CTRL/Q).

If there are no messages for any program, a message is displayed indicating there are none.

13.4.3 Option MA

Option MA displays the contents of all messages in the queue.

Typical Option MA response:

NO. MESSAGE FOR PROGRAM	DATE	SIZE
1 TST .SAV THIS IS A MESSAGE	30-MAR-83	24
2 TST2 .SAV THIS IS MESSAGE TWO	30-MAR-83	88
3 TST3 .SAV THIS IS MESSAGE THREE	30-MAR-83	24

ENTER OPTION (H,M,MA,MX,MKX,X):

The first 80 characters of the message is displayed. If more than 11 messages are in the queue, the first 11 will appear and, after a short pause, the rest of the messages will be scrolled onto the screen. The display can be stopped using the SCROLL/NO SCROLL key (or CTRL/S and CTRL/Q).

13.4.4 Option Mx

Option Mx displays the contents of a selected message in the same format as Option M. The message is specified by a numeric value for x and can be one or two characters. Use the M or MA option to determine the message number.

Typical Option Mx response:

NO. MESSAGE FOR PROGRAM	DATE	SIZE
4 TST .SAV THIS IS MESSAGE FOUR	30-MAR-83	24

ENTER OPTION (H,M,MA,MX,MKX,X):

If the message number is not valid (that is, shown with the M option), a message is displayed indicating the number is invalid.

13.4.5 Option MKx

Option MKx deletes (kills) a message selected by the specified value of x which can be one or two characters. The message number is obtained by using Option M or MA. After deletion of the message, the Option M display is presented showing the remaining contents of the message queue.

13.4.6 Option X

This is the exit option. It allows control to return to the operating system. When option X is selected, the operating system regains control of the terminal and displays the operating system prompt.

CHAPTER 14

REDUCE

14.1 INTRODUCTION

REDUCE is a time-shared utility program that decreases the size of a file linked at a base address of 100000 (octal) (32 K bytes). A file so linked includes 63 unused blocks. In a small system, especially those using diskettes, this wasted space can become excessive. REDUCE accepts an input file, reads it, and then copies the file, minus the 63 unused blocks. Finally, it deletes the input file. For overlaid files, the relative block number in the overlay handler tables is also modified.

14.1.1 Characteristics

REDUCE requirements and limitations are:

- Requires 4 K bytes of memory.
- Recognizes file-structured devices only.
- Supported by RT-11 Version 3 or later.
- Recognizes certain wildcard constructions for input file specification.

14.1.2 Chapter Organization

The remainder of this chapter is comprised of three sections. The first, Section 14.2, REDUCE Options, is a discussion of the options available with REDUCE. The second, Section 14.3, Using REDUCE, describes input file conventions and how to run REDUCE. The last section, Section 14.4, REDUCE Examples, presents several examples illustrating mode, option, and wildcard use.

14.1.3 Error Messages

Error messages associated with REDUCE are documented in the *CTS-300 System Message Manual*.

14.2 REDUCE OPTIONS

There are three modes in which REDUCE can operate: query, no query, and version number. These modes result from options selected when file(s) are specified to be reduced. There are two options. Modes and options are discussed below. Because REDUCE is a relatively simple program, selection and format for mode, option, and wildcards are shown using examples in Section 14.4.

14.2.1 Query Mode

This is the default mode; no option is specified. Each file that satisfies the stated file specification(s) is listed for you to decide whether to reduce it. You respond with Y if you want the file reduced, or with N (or carriage return) if you do not want the file reduced. If you attempt to reduce a file that was previously reduced or to reduce a file that was not linked for a base of 100000, a message will be displayed indicating that REDUCE has ignored that file.

14.2.2 /N Option (No Query)

The /N option suppresses the individual file queries present in the query mode. This is the no query mode. Processing proceeds on all the files that meet the file specification(s). This option provides for faster processing of your files. Messages are displayed, as in the query mode, indicating a previously reduced file or a file not linked for a base of 100000.

14.2.3 /V Option (Version Number)

The /V option causes the REDUCE utility to display its current version number. This is the version number mode.

14.3 USING REDUCE

14.3.1 Conventions

Input files

- REDUCE operates only on files that have been linked for a base address of 100000 (octal). Upon receipt of an input file specification, REDUCE searches the header block for the correct link address, and ensures that the file has not already been reduced.
- More than one input file can be specified.
- Asterisk wildcards can be used to specify files with a common file name or a common extension.

Output files

The output file(s) resulting from reduction has the same name as the input file(s).

14.3.2 Running REDUCE

REDUCE is supplied as an executable file which is run like any other program in response to the system prompt:

```
.R (RU) REDUCE
```

REDUCE responds with an asterisk prompt for the input file(s).

The general form of file specification and option selection is:

filespec/n

where:

filespec is the file specification of the file(s) to be reduced.

- If more than one file is to be reduced, the file specifications are separated by commas.
- REDUCE assumes a default extension of .TSD.
- REDUCE recognizes limited use of asterisk wildcards used to specify files with a common file name and/or extension. The use of single character wildcards or the use of the asterisk wildcard in combination with other characters is not supported.

/n is the option specifier with n being either an N for no query or a V for a version number.

14.4 REDUCE EXAMPLES

The modes and options available with REDUCE are illustrated in the following subsections.

14.4.1 Query Mode

To reduce the two files MYFIL and YURFIL in query mode, enter the following in response to the asterisk prompt:

```
*MYFIL,YURFIL
```

The result will be the query:

```
DK:MYFIL.TSD?
```

The file is on the default device and exists with a .TSD extension. Respond with Y or N. The next line will be:

```
DK:YURFIL.TSD?
```

The same query and possible responses exist as did for the first file.

14.4.2 No Query Mode

To reduce the two files used in the query mode example but without the query, the response to the asterisk prompt is:

```
*MYFIL,YURFIL/N
```

to which the only response would be the asterisk prompt indicating that the files have been reduced.

14.4.3 Wildcards

To reduce all files with a .TSD extension on the default device, the response to the prompt is:

```
*.TSD
```

To reduce all files with file name TEST on the default device, the response to the prompt is:

```
*TEST.*
```

To reduce all files on device DL1: the response to the prompt is:

```
*DL1:*.*
```

In all cases the only response would be the asterisk prompt indicating that the files have been reduced. That is, use of wildcards results in the no query mode.

14.4.4 Version Number Mode

To obtain the version number of the REDUCE utility, the response to the asterisk prompt is:

```
*/V
```

The response is

```
REDUCE UTILITY VAnn-nn
```

where:

nn-nn is the current version number.

CHAPTER 15

MESSAGE UPDATE UTILITY

15.1 INTRODUCTION

A message file (ERMSG.TXT) contains messages for the CTS-300 run-time system and supplied utilities. When required, the run-time system or a system utility will obtain a message from this file and output it to a terminal. The Message Update Utility (MSGUTL) provides maintenance capability for these messages.

This message file and the associated method of access is not intended to be utilized to handle messages in programs written by the CTS-300 user. It is designed to be a tool for CTS-300 system and utility program messages.

15.1.1 Features

MSGUTL allows you to do the following:

- Add and/or update text in the message file.
- Delete the messages for a specified program.
- Use a terminal or a command file to input changes.
- List a segment on either a terminal or a line printer.

15.1.2 Limitations

MSGUTL does not support certain operations:

- You cannot delete individual messages.
- MSGUTL runs only in a single-user environment.
- Dialog messages for DKED, DECFORM, and the printer spoolers are included in the message file. Dialog for other programs is not included because the programs do not access the message file for dialog.
- Because the message facility is not in place when they occur, certain kinds of messages cannot be included in the message file. These are messages originating at system initialization time (single-user and time-shared) and a few messages associated with DICOMP, FOCOMP, REDUCE, LPTSP1, and SORT.
- Messages for MSGUTL itself are not included in the file.

The message file has the following limitations:

- The message file can reside on any device (physical or logical). Specification of the device is done during CTSGEN. The run-time system looks for the file on the specified device. If not found on the specified device, it looks for the file first on DK: then on SY:.
- The default location for the error messages for the supplied utilities (i.e., the DIBOL compiler, MSGUTL.SAV, SORT.SAV, etc.) is logical device ERR. If the messages are not on this device, a search is first made of DK: followed by a search of SY:.

NOTE

Only MSGUTL can be used to modify the message file. Attempts to use a text editor will make the file unusable.

15.1.3 Chapter Organization

The remainder of this chapter is organized in three sections. Section 15.2, Message File Structure, introduces the message file. Section 15.3, Updating Messages, explains how to use MSGUTL to modify the message file. Section 15.4, Examples, contains examples illustrating how MSGUTL is used. Both terminal input and control file input are shown.

15.2 MESSAGE FILE STRUCTURE

Before using the Message Update Utility you must be familiar with the message file, ERMSG.TXT. The first record of the message file is a binary header. The remainder is basically a DIBOL sequential file consisting of 64-character records. Each record contains a message. For the purposes of message identification the file is divided into segments. A segment is assigned to each program or utility which is to have access to the message file. Each segment consists of a variable number of records depending on the number of messages.

The number of segments is limited to 20. The maximum number of entries (messages) in a segment is 999. Segments are stored contiguously to minimize storage requirements, and in practice the size of the file is approximately 100 blocks. However, since each entry is a 64-character fixed length record, the maximum size the message file could become is approximately 2500 blocks.

15.2.1 Segments

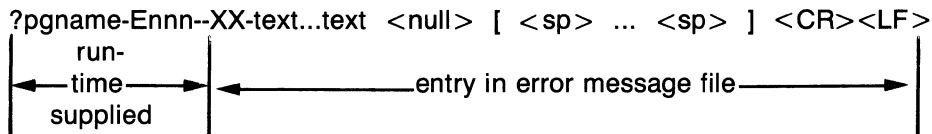
Each segment in the message file is assigned to a specific program or CTS-300 utility. Segment assignment for the CTS-300 Version 7 release is:

Seg. Program/Utility	Seg. Program/Utility
1 DIBOL Errors	11 SORT
2 CTSRTS	12 SORTG
3 DDT	13 SORTM
4 DECFORM	14 Spoolers
5 DKED	15 STATUS
6 FB Communication	16 DICOMP
7 ISMUTL	17 FOCOMP
8 PRINTU	18 DECTYPE
9 QUILL	19 CTSDCL
10 REDUCE	20 RESERVED

Segment size is determined by the number of messages (records) that are entered for that segment. As a message is added to a segment, it is assigned a sequential record number by MSGUTL. Thus each message has an associated number. Messages are accessed on the basis of this number.

15.2.2 Messages

Messages consist of four parts: program identifier, error number, type, and text. The program identifier and the error number are supplied by the error handler which is part of the DIBOL run-time system. The message type and text are stored in the message file. The message file entry size of 64 characters includes four characters for the message type designator and delimiting dashes, a null character, and two characters for the carriage return/line feed pair. This leaves 57 characters for the message text. Messages with less than 57 characters of text are filled out with spaces. The format of the entire message is summarized below:



where:

?pname-Ennn-

is supplied by the message handler and identifies the program in which the error occurred or which requested the message and the message number.

-XX-

is the message type. The first and fourth characters must be dashes. It is part of the entry in the file and can be modified using MSGUTL. However, the only need for modification would be for a language other than English.

The allowable types are:

TYPE	MEANING
E	error
W	warning
F	fatal
NT	not trappable
T	trappable
I	informational/instructional
D	dialog
L	list
H	help

If type is a single letter, the first X must be replaced by a dash.

text is the message text

<null> is a null character inserted by MSGUTL when the file is built.

<sp> ... <sp>
are spaces inserted by MSGUTL to make the record 64 characters long.

<CR> <LF>
is the normal record terminator required of any DIBOL record. It is inserted by MSGUTL when the file is built.

15.3 UPDATING MESSAGE FILE RECORDS

The message file was created to facilitate message change. A terminal can be used in an interactive mode with the commands and options available in MSGUTL. A command file can also be used to supply this information.

NOTE

Do not attempt to edit the message file with a text editor. Some characters are stored in binary form. While they can be read by an editor, the editor can modify these binary values so they are no longer usable by either MSGUTL or the run-time system.

This section discusses how to run MSGUTL, how to select modes and issue commands at a terminal, and how to use a control file as input.

15.3.1 Running the Utility

The utility is run (single-user system only) with the following command:

```
.R MSGUTL
```

The response is:

```
MESSAGE UPDATE UTILITY
```

Input will be:

- (1) From the terminal
- (2) From a control file

Enter option:

At this point the method of input is selected. MSGUTL accepts input either from a terminal or from a control file as described in Sections 15.3.3 and 15.3.6. Once running, various modes of operation and keyboard commands are available for file manipulation.

15.3.2 Update Utility Error Messages

Messages are generated by the Message Update Utility as a result of invalid input, an attempt to perform an invalid function, or the inability of the system to perform a given function. All MSGUTL messages are documented in the *CTS-300 System Message Manual*.

15.3.3 Terminal Input

An interactive terminal session is selected by entering a 1 in response to the input selection display seen upon running the utility. An example of a terminal session is shown in Section 15.4.1.

Aside from the input selection display, shown in Section 15.3.1, there are two displays frequently used by MSGUTL. The first is a menu of the available operating modes (mode menu). The second is a directory of the message file (MSGUTL directory).

Mode menu display:

MESSAGE UPDATE UTILITY

Modes:

- (1) ADD a message to a segment
- (2) MODIFY a message in a segment
- (3) DELETE a whole segment
- (4) LIST a segment on TT:
- (5) LIST a segment on LP:
- (6) EXIT this session with changes made to file
- (7) ABORT this session without changes made to file

Enter option:

MSGUTL directory display:

SEGMENT	MESSAGE COUNT	SEGMENT	MESSAGE COUNT
(1) DIBOL Errors	84	(11) SORT	41
(2) CTSRTS	12	(12) SORTG	28
(3) DDT	5	(13) SORTM	9
(4) DECFORM	69	(14) Spoolers	151
(5) DKED	47	(15) STATUS	8
(6) FB Communication	10	(16) DICOMP	58
(7) ISMUTL	22	(17) FOCOMP	79
(8) PRINTU	33	(18) DECTYPE	<EMPTY>
(9) QUILL	<EMPTY>	(19) CTSDCL	26
(10) REDUCE	18	(20) **RESERVED**	<EMPTY>

Delete character key and the CTRL/U (delete line) function are operational while MSGUTL is running.

At various times MSGUTL will briefly display a message indicating that the system is waiting for a temporary file to be opened or a message may appear requesting you wait while the next operation is being set up.

The following sections discuss the modes of operation and keyboard commands.

15.3.4 Modes of Operation

When MSGUTL is run, the function it performs is dependent upon your selection of a mode of operation. After you select terminal input, the mode menu is displayed. Each of the seven operating modes are discussed in the following subsections.

Section 15.3.5 describes in detail the keyboard commands referred to in this section.

15.3.4.1 ADD a message (Mode 1) — Displays the MSGUTL directory followed by a prompt for segment number.

When the segment number is entered, the first unassigned message number in the segment is displayed. The asterisk prompt indicates a request for message input. Type the message (including the type identifier and dashes) followed by a carriage return.

Messages can be modified while in the ADD mode by using the ESC/M command. Once the modification is complete, return to the end of the segment with the ESC/R keyboard command to add more messages.

Use the ESC/E command to exit the ADD mode.

15.3.4.2 MODIFY a message (Mode 2) — Displays the MSGUTL directory followed by a prompt for segment number. Your response results in a prompt for the number of the message you want to modify. This message prompt is repeated after each message is changed and the carriage return entered.

Use the ESC/E command to exit the MODIFY mode.

15.3.4.3 DELETE a segment (Mode 3) — Displays the MSGUTL directory followed by a prompt for segment number. The segment you specify is deleted when you exit from MSGUTL via the EXIT mode (Mode 6); if you decide not to delete the segment you can cancel the deletion by using the ABORT mode (Mode 7).

Use the ESC/E command to exit the DELETE mode.

15.3.4.4 LIST a segment on TT: (Mode 4) — Displays the MSGUTL directory followed by a prompt for the segment number. When the number is entered, all the messages in that segment are displayed on the terminal. When the list is completed you are again prompted for a segment number.

Use the ESC/E command to exit this LIST mode.

15.3.4.5 LIST a segment on LP: (Mode 5) — Displays the MSGUTL directory followed by a prompt for the segment number and then lists the messages in that segment on the terminal, and prints the list on the line printer assigned to LP:. When printing is complete you are again prompted for a segment number.

Use the ESC/E command to exit this LIST mode.

15.3.4.6 EXIT this session with changes made (Mode 6) — Exits the MSGUTL utility and makes permanent the changes (additions, modifications, deletions) made to the message file. Return is to the RT-11 monitor prompt.

15.3.4.7 ABORT this session without changes made (Mode 7) — Exits the MSGUTL utility leaving the file as it was before the utility was run. Return is to the RT-11 monitor prompt.

15.3.5 Keyboard Commands

MSGUTL keyboard commands are used once a mode has been chosen. MSGUTL keyboard commands allow you to position yourself at a different message, list messages in the segment, display a help frame, or indicate a message or segment that is to be changed.

MSGUTL commands are formed using the ESCAPE key in combination with other keys. The ESCAPE key is pressed first followed by the other key.

On VT52 and VT100 terminals the arrow keys can be used (as shown) in place of some of the escape key combinations.

15.3.5.1 [ESC/A] (up arrow)

Valid modes of operation: ADD, MODIFY

Can be used in response to either the input selection display or to a message number when in a mode dealing with individual messages.

This keyboard command moves you backward through the segment one message at a time. As the command is issued, successively lower numbered messages are displayed. Below the message is the message number followed by an asterisk with the cursor positioned to the right of the asterisk. At this point, the message can be changed by entering new text or another command can be issued.

If you are at message number 1, this commands causes the audible alarm to sound indicating that you cannot back up any further.

15.3.5.2 [ESC/B] (down arrow)

Valid modes of operation: ADD, MODIFY

Can be used in response to either the input selection display or to a message number when in a mode dealing with individual messages.

This keyboard command moves you forward through the segment one message at a time. As the command is issued, successively higher numbered messages are displayed. Below the message is the message number followed by an asterisk with the cursor positioned to the right of the asterisk. At this point, the message can be changed by entering new text or another command can be issued.

If you are at the last message in the segment, this command causes the audible alarm to sound indicating that you cannot advance any further.

15.3.5.3 [ESC/D] (left arrow)

Valid modes of operation: MODIFY, DELETE, during mode selection

Can be used in response to any mode display.

This keyboard command displays the MSGUTL directory followed by the last line displayed prior to issuance of the ESC/D command.

15.3.5.4 [ESC/E]

Valid modes of operation: ADD, MODIFY, DELETE

This keyboard command causes an exit from the present mode and presents the mode menu display.

This command is the normal way to return to the mode menu.

15.3.5.5 [ESC/H]

Valid modes of operation: ADD, MODIFY, DELETE, during mode selection

This keyboard command causes the MSGUTL help frame to be displayed. The help frame lists the keyboard commands and their basic functions:

[ESC] A or [UP ARROW]	Move backward through the segment
[ESC] B or [DOWN ARROW]	Move forward through the segment
[ESC] D or [←]	Directory of Segments & Message Counts
[ESC] E	Exit the present MODE
		Returns to the "Mode menu" display
[ESC] H	Displays the help frame
[ESC] L	List all messages in present segment on TT:
[ESC] M	Move to a different message in this segment
[ESC] R[ADD MODE ONLY].....	Returns to next message to be added after an UP or DOWN ARROW, [ESC]A, [ESC]B or [ESC]M
[ESC] S	Move to a different segment

NOTE: Messages are updated by typing in an alpha character followed by a <RETURN> when responding to the prompt *

NOTE: [UP ARROW], [DOWN ARROW], and [←] apply only to VT52 and VT100 terminals

The help frame is followed by the last line displayed before the ESC/H command was issued.

15.3.5.6 [ESC/L]

Valid modes of operation: ADD, MODIFY, DELETE

This keyboard command causes the messages for the current segment to be listed in numerical order on the terminal in the same format used with Mode 4. After the list, the last line displayed before this command was issued is re-displayed.

15.3.5.7 [ESC/M]

Valid modes of operation: ADD, MODIFY

This keyboard command is used to initiate a message change. The response is a prompt for the message number. The message corresponding to the number entered is displayed and the cursor positioned to the right of the asterisk to enable text change.

15.3.5.8 [ESC/R]

Valid mode of operation: ADD

This command is used only after you have moved away from the end of the segment as a result of an ESC/M or ESC/A sequence.

This keyboard command returns you to the end of the segment to add another message.

15.3.5.9 [ESC/S]

Valid modes of operation: ADD, MODIFY

This keyboard command allows you to move to another segment while remaining in the present mode. The response is a prompt for the segment number followed by a prompt appropriate for the mode which you are in.

15.3.6 Control File Input

Under some circumstances it is more convenient to use a control file to supply the necessary mode selection, message text insertion, and keyboard commands. The command file method lends itself to such operations as the fast generation of a message file consisting of messages translated into a language other than English.

Two examples of a control file used as input are shown in Section 15.4.2. Details of the control file input method are contained in that section.

15.4 EXAMPLES

15.4.1 Terminal Input Operation

In this example four modes and various keyboard commands are illustrated. An explanation precedes each separate operation.

The example:

First, simple messages are added to segment 20 (previously empty) using mode 1.

```
.R MSGUTL
```

MESSAGE UPDATE UTILITY

Input will be:

- (1) From the terminal
- (2) From a control file

Enter option: 1

MESSAGE UPDATE UTILITY

Options:

- (1) ADD a message to a segment
- (2) MODIFY a message in a segment
- (3) DELETE a whole segment
- (4) LIST a segment on TT:
- (5) LIST a segment on LP:
- (6) EXIT this session with changes made to file
- (7) ABORT this session without changes made to file

Enter option: 1

SEGMENT	MESSAGE COUNT	SEGMENT	MESSAGE COUNT
(1) DIBOL Errors.....	84	(11) SORT.....	41
(2) CTSRTS.....	12	(12) SORTG.....	28
(3) DDT.....	5	(13) SORTM.....	9
(4) DECFORM.....	69	(14) Spoolers.....	151
(5) DKED.....	47	(15) STATUS.....	8
(6) FB Communication.....	10	(16) DICOMP.....	58
(7) ISMUTL.....	22	(17) FOCOMP.....	79
(8) PRINTU.....	33	(18) DECTYPE.....	<EMPTY>
(9) QUILL.....	<EMPTY>	(19) CTSDCL.....	26
(10) REDUCE.....	18	(20) **RESERVED**.....	<EMPTY>

ADD MODE

Enter Segment Number: 20

Please wait.....

SEGMENT 20:

1*--I-This is message one <CR>
2*--I-This is message two <CR>
3*--I-This is message three <CR>
4*--I-This is message four <CR>
5*--I-This is message five <CR>
6*--I-This is message six <CR>
7*--I-This is message seven <CR>
8*--I-This is message eight <CR>
9*--I-This is message nine <CR>
10*--I-This is message ten <CR>
11* <ESC/D>

At this point an ESC/D command was issued to check the size of the segment. Notice that segment 20 now has a count of 10 whereas before it was empty.

SEGMENT	MESSAGE COUNT	SEGMENT	MESSAGE COUNT
(1) DIBOL Errors.....	84	(11) SORT.....	41
(2) CTSRTS.....	12	(12) SORTG.....	28
(3) DDT.....	5	(13) SORTM.....	9
(4) DECFORM.....	69	(14) Spoolers.....	151
(5) DKED.....	47	(15) STATUS.....	8
(6) FB Communication.....	10	(16) DICOMP.....	58
(7) ISMUTL.....	22	(17) FOCOMP.....	79
(8) PRINTU.....	33	(18) DECTYPE.....	<EMPTY>
(9) QUILL.....	<EMPTY>	(19) CTSDCL.....	26
(10) REDUCE.....	18	(20) **RESERVED**.....	10

11* <ESC/L>

More messages could be added but an ESC/L command was used to display a list of the messages.

SEGMENT 20:

MESSAGE NUMBER	MESSAGE TEXT
1	--I-This is message one
2	--I-This is message two
3	--I-This is message three
4	--I-This is message four
5	--I-This is message five
6	--I-This is message six
7	--I-This is message seven
8	--I-This is message eight
9	--I-This is message nine
10	--I-This is message ten

11* <ESC/M>

When this list was displayed, it was noticed for the first time that one of the messages was misspelled. An ESC/M command was used to select this particular message (5) and new text is entered.

```
Enter message number: 5
--I-This is message five
5*--I-This is message five <CR>
Enter message number: <ESC/R>
```

Because more messages are to be added, an ESC/R command was used to return to the end of the segment. Continuing in response to the 11* displayed following the ESC/R, more messages are entered.

```
11*--I-message 11
12*--I-message 12
13*--I-message 13
14*--I-message 14
15*--I-message 15
16* <ESC/E>
```

Mode 2 will now be used to modify some of the messages just inserted in segment 20. The use of position commands (ESC/A and ESC/B) will be shown. The ESC/E was used to display the mode menu so we could go to the MODIFY mode from the ADD mode.

MESSAGE UPDATE UTILITY

Options:

- (1) ADD a message to a segment
- (2) MODIFY a message in a segment
- (3) DELETE a whole segment
- (4) LIST a segment on TT:
- (5) LIST a segment on LP:
- (6) EXIT this session with changes made to file
- (7) ABORT this session with out changes made to file

Enter option: 2

SEGMENT	MESSAGE COUNT	SEGMENT	MESSAGE COUNT
(1) DIBOL Errors.....	84	(11) SORT.....	41
(2) CTSRTS.....	12	(12) SORTG.....	28
(3) DDT.....	5	(13) SORTM.....	9
(4) DECFORM.....	69	(14) Spoolers.....	151
(5) DKED.....	47	(15) STATUS.....	8
(6) FB Communication.....	10	(16) DICOMP.....	58
(7) ISMUTL.....	22	(17) FOCOMP.....	79
(8) PRINTU.....	33	(18) DECTYPE.....	<EMPTY>
(9) QUILL.....	<EMPTY>	(19) CTSDCL.....	26
(10) REDUCE.....	18	(20) **RESERVED**.....	15

MODIFY MODE

Enter Segment Number: 20

SEGMENT 20:

Enter Message Number: 13

--I-message 13

13* --I-This is message thirteen <CR>

Enter Message Number: 12

--I-message 12

12* --I-This is message twelve <CR>

Enter Message Number: <ESC/A> (or up arrow)

--I-message 11

11* <ESC/A> (or up arrow)

--I-This is message ten

10* <ESC/B> (or down arrow)

--I-message 11

11* --I-This is message eleven <CR>

Enter Message Number: <ESC/B> (or down arrow)

--I-This is message twelve

12* <ESC/B> (or down arrow)

--I-This is message thirteen

13* <ESC/B> (or down arrow)

--I-message 14

14* --I-This is message fourteen <CR>

Enter Message Number: <ESC/B> (or down arrow)

--I-message 15

15* --I-This is message fifteen <CR>

Enter Message Number: <ESC/B> (or down arrow) (alarm will sound because there are no messages beyond 15)

Enter Message Number: <ESC/E>

Because these are all the modifications wanted, the ESC/E command was used to return to the mode menu display. Mode 3 will be used to delete segment 20 which returns to the empty state followed by mode 6 to exit the utility.

MESSAGE UPDATE UTILITY

Options:

- (1) ADD a message to a segment
- (2) MODIFY a message in a segment
- (3) DELETE a whole segment
- (4) LIST a segment on TT:
- (5) LIST a segment on LP:
- (6) EXIT this session with changes made to file
- (7) ABORT this session with out changes made to file

Enter option: 3

SEGMENT	MESSAGE COUNT	SEGMENT	MESSAGE COUNT
(1) DIBOL Errors	84	(11) SORT	41
(2) CTSRTS	12	(12) SORTG	28
(3) DDT	5	(13) SORTM	9
(4) DECFORM	69	(14) Spoolers	151
(5) DKED	47	(15) STATUS	8
(6) FB Communication	10	(16) DICOMP	58
(7) ISMUTL	22	(17) FOCOMP	79
(8) PRINTU	33	(18) DECTYPE	<EMPTY>
(9) QUILL	<EMPTY>	(19) CTSDCL	26
(10) REDUCE	18	(20) **RESERVED**	15

DELETE MODE

Enter Segment Number: 20

SEGMENT	MESSAGE COUNT	SEGMENT	MESSAGE COUNT
(1) DIBOL Errors	84	(11) SORT	41
(2) CTSRTS	12	(12) SORTG	28
(3) DDT	5	(13) SORTM	9
(4) DECFORM	69	(14) Spoolers	151
(5) DKED	47	(15) STATUS	8
(6) FB Communication	10	(16) DICOMP	58
(7) ISMUTL	22	(17) FOCOMP	79
(8) PRINTU	33	(18) DECTYPE	<EMPTY>
(9) QUILL	<EMPTY>	(19) CTSDCL	26
(10) REDUCE	18	(20) **RESERVED**	<EMPTY>

DELETE MODE

Enter Segment Number: <ESC/E>

MESSAGE UPDATE UTILITY

Options:

- (1) ADD a message to a segment
- (2) MODIFY a message in a segment
- (3) DELETE a whole segment
- (4) LIST a segment on TT:
- (5) LIST a segment on LP:
- (6) EXIT this session with changes made to file
- (7) ABORT this session with out changes made to file

Enter option: 6

Please wait

.(the RT-11 monitor prompt)

15.4.2 Control File Input

This section shows examples of a control file being used to supply the input to MSGUTL.

A text editor is used to create a file with the modes and keyboard commands as line items. All lines will be terminated with the carriage return.

Special Keys:

- The “@” character is used in place of the <ESC> key to identify commands. It cannot be used as the first character of a message.
- The “!” character is used to identify comments to the file.

If illegal input is encountered in the control file, a message is displayed that states the control file line number and the fact that the wrong data type was encountered.

We will now create a control file called ADDS.MSG using the DKED editor. The file will achieve the same result (but without the misspelling) as the first series of added records for the session shown for terminal input:

```
*DKED
*ADDS.MSG =
!Mode 1 selection:
1
!Segment 20 selection:
20
!Messages follow:
--!-This is message one
--!-This is message two
--!-This is message three
--!-This is message four
--!-This is message five
--!-This is message six
--!-This is message seven
--!-This is message eight
--!-This is message nine
--!-This is message ten
!Exit to the mode menu display:
@E
!Exit MSGUTL:
6
!DKED exit:
<GOLD/COMMAND>
EXIT
```

To run the utility from the control file:

```
.R MSGUTL
```

```
MESSAGE UPDATE UTILITY
```

Input will be:

- (1) From the terminal
- (2) From a control file

Enter option: 2

Enter file name: ADDS.MSG

All output is displayed on the terminal as the command file is executed.

A control file to modify a file segment could look like the following:

Assume segment 20 has ten messages each message being simply the number of the message (message one is 1, etc.). A control file, MODIFY.MSG, to change messages 1, 5, and 6 could be:

```
*DKED
*MODIFY.MSG=
2
20
1
--I-This is message number one in segment 20
5
--I-This is message number five in segment 20
@B
--I-This is message number six in segment 20
@E
6
<GOLD/COMMAND>
EXIT
```

The result (in list format) would be:

MESSAGE NUMBER	MESSAGE TEXT
1	--I-This is message number one in segment 20
2	:2
3	:3
4	:4
5	--I-This is message number five in segment 20
6	--I-This is message number six in segment 20
7	:7
8	:8
9	:9
10	:10

An indirect command file could be created that would eliminate the first three steps shown above for the example using the ADDS.MSG file as input. Such a file, using DKED, would look like:

```
*DKED
*ADDS.COM =
R MSGUTL
2
ADDS.MSG
<GOLD/COMMAND>
EXIT
```

The command to execute the file to add messages would then be:

```
.@ADDS
```


APPENDIX A

ISAM FILE CHARACTERISTICS

A.1 INTRODUCTION

This appendix introduces CTS-300 ISAM and its terminology and explains the internal structure in the detail necessary for you to be able to build and use an ISAM file. Chapter 10 describes ISMUTL, the utility used to build and maintain ISAM files.

A.1.1 Features

In general, ISAM files offer several advantages over random or sequential files:

- simplified record storage and retrieval
- greater file access speed
- ability to delete records
- more easily designed and modified to fit the growth requirements of the application

Specifically, ISAM files are better suited to some applications than others.

ISAM files are particularly well suited for:

- record access and update
- limited addition of new records to the file

ISAM files are not well suited for:

- random access of records when the file is in an overflow condition (see Sections A.2.3, A.3.1.2, A.3.2.5, and A.3.5)
- building a new file if the records are going to be randomly selected for storage

A.1.2 Appendix Organization

This appendix is comprised of two sections. First is Section A.2, ISAM Basics, for users who are not familiar with ISAM concepts and terminology. Section A.3, ISAM Internals, discusses the internal structure of an ISAM file and explains the relationships that exist within the file as well as the factors that must be considered when designing an ISAM file.

A.2 ISAM BASICS

The basic concepts and terminology of ISAM are presented in this section. It is a brief, overall description of an ISAM file. The purpose is to provide background for detailed discussions in the following section. The Indexed Sequential Access Method (ISAM) is a means of organizing data for storage and subsequent access. An ISAM file is composed of two major sections: the data section and the index section.

A.2.1 Data Section

Records are assigned, in ascending order, to a specific location in an ISAM file, called the data section, which is comprised of data files. A user-identified field within the record determines where that record is stored in a data file within the data section. This field is called the record key. In CTS-300 ISAM, there is only one such key field per record. Records are sequentially stored by this key in data groups of constant size. These data groups are organized to form as many as seven data files. These data files constitute the data section.

A.2.2 Index Section

The greatest record key within each data group is entered, along with the address of its associated data group, into an index. The index is contained in the index section, often called the index file. This index section (index file) is the second major division of an ISAM file. By searching this index for a key match, the group which contains the desired record may be quickly accessed. Each record in the group is then searched sequentially for a key match, and the desired record is obtained.

A.2.3 Handling Added Records

Suppose you want to store a record that, because of its key value, belongs in a given group but which cannot be placed there because the group is full. One solution would be to leave a few empty record spaces in each group when the file is first constructed and the initial data is loaded. When this is done, the number of spaces left empty is called the load exclusion factor. But what happens when even these spaces in a given group are filled? A way to handle this is to set aside an area somewhere within the total ISAM file to place the new records. This is called the overflow area, and it physically precedes the index in the index section. This overflow area is organized in groups identical to data file groups but, as will be explained later, access is different.

There is one remaining question: How do you add records which have greater key values than those of any existing group? These new greater key records cannot go into overflow, because they do not logically belong in any existing group. There is no such group, because when an ISAM file is created from a given input file, the ISAM file size is determined by the initial space requirements of the input file. The solution is to provide space at the end of the data area, in addition to the known initial space requirement. This area is known as the append area.

A.2.4 Summary of ISAM Basics

In summary, there are two basic parts to an ISAM file: the index section and the data section. The index section is by far the smaller of the two and contains the overflow record groups and the index to the data portion of the file. A search for data is made via the index which points to a data group which is, upon access, searched sequentially for the desired record. Added records of intermediate key value go into open record spaces, load exclusion record spaces or, if the group is full, into the overflow area. Records with keys greater than the current key range are placed in the append area. The structure of an ISAM file is illustrated in Figure A-1.

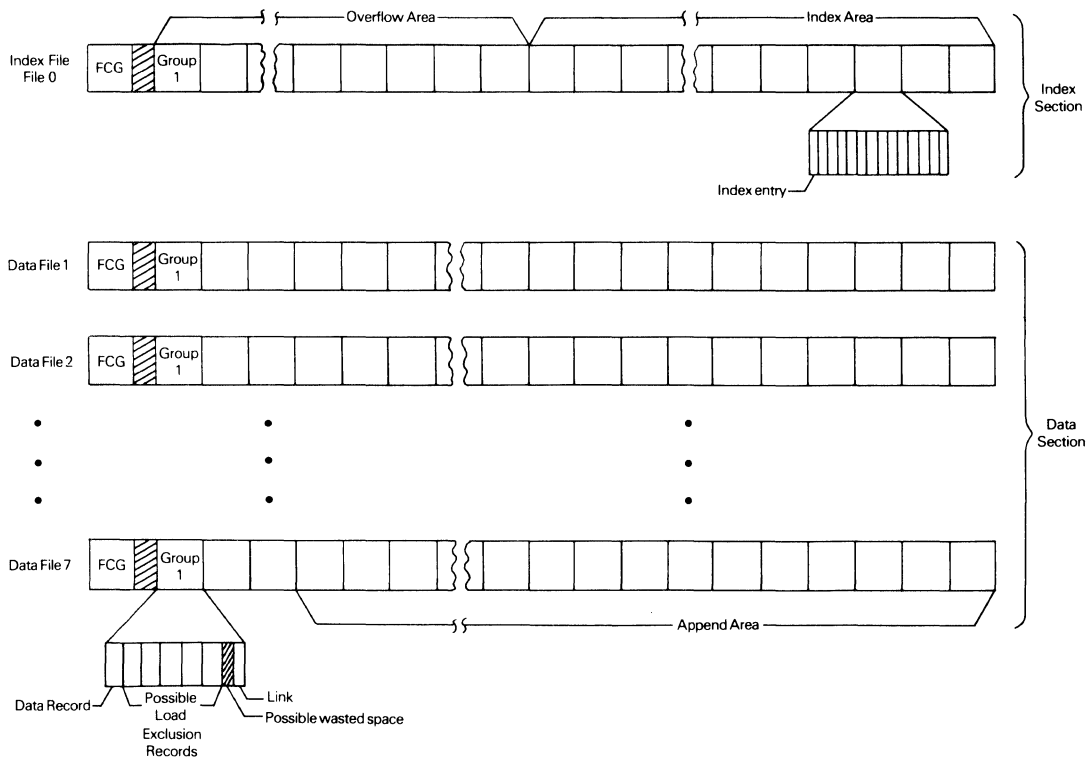


Figure A-1 ISAM File Overview

A.3 ISAM INTERNALS

The previous section presented a simplified picture of an ISAM file. However, a more detailed understanding is necessary for the ISAM user in order to respond to questions asked by the ISAM utility (see Chapter 10) that will direct that utility to construct the index, overflow, and append areas. To build a file that provides maximum efficiency in terms of access speed and storage requirements requires a thorough and complete understanding of both the ISAM file itself and the particular requirements dictated by the data. This section discusses the file structure in detail and the interrelationships between the various parts of the file.

A.3.1 Detailed Structure

A.3.1.1 Data Files — Data files are discussed first, because the overflow area is constructed like a data file, and also because the index and overflow area are more meaningful after the data file is understood.

As stated before, there may be up to seven data files (or logical volumes) in an ISAM file. These are labeled 1 through 7. Logical volumes are RT-11 files which may be located on the same physical volume (a disk, for example) or on different physical volumes. All access to the file is done using the name assigned to the index file. The main reason for multiple data files is to allow division of the total ISAM data file over the system resources.

Each of the data files in an ISAM file may be a different size but all are constructed the same. Each file is segmented into sequentially numbered groups of equal size. The first group (group 0) of each data file is reserved for system use and contains only 132 bytes of data that are meaningful, regardless of group size. This first group is called the File Control Group (FCG). FCGs are created at the beginning of each data file and at the beginning of the index file when the ISAM file is created. As the ISAM file changes, only the FCG in the index file is updated. More information about the FCG is contained in the index file discussion. All remaining groups in the data file are data groups.

Each data group consists of a record area and, at the end of the group, an area containing linking information to the next logical data group. Within the record area are the data records which are of equal length. Data groups eventually become filled as records are stored.

When each group is filled, the records are ordered so that the last record in the group has the greatest key. This key is then inserted in a preassigned position in the index file. Thus each data group has a corresponding index entry. Index space is set aside when the ISAM file is originally built, with one index entry being allocated per data group.

The last four bytes of each group are reserved for linking and are not accessible to the user. The link information consists of one byte indicating the number of presently valid records in the group; one byte for the file number of the next logical data file; and two bytes for the number of the next logical data group within that next logical data file. Until overflow occurs, a link will always point to the next data group in the present file or to the first data group in the next data file. When overflow occurs, the link points to a group in the overflow area. The groups in overflow are identical in structure, including link structure, to those in the data file. The link in the overflow group points back to the original data file if the number of records requires only one overflow group; or if more than one overflow group were required to contain the overflow records, the link in the last overflow group would point back to the original data file.

The append area is simply an extension of the data section, resulting from the need for more storage area than is indicated by the input file size. Since the append area is a part of the data section, added records in the append area may also cause overflow.

The choice of group size depends on many factors and these are discussed in Section A.3.2.

Figure A-2 illustrates the concepts of data group organization discussed above using a group that is 512 bytes in size with five 100-byte records, two of which are identified as load exclusion.

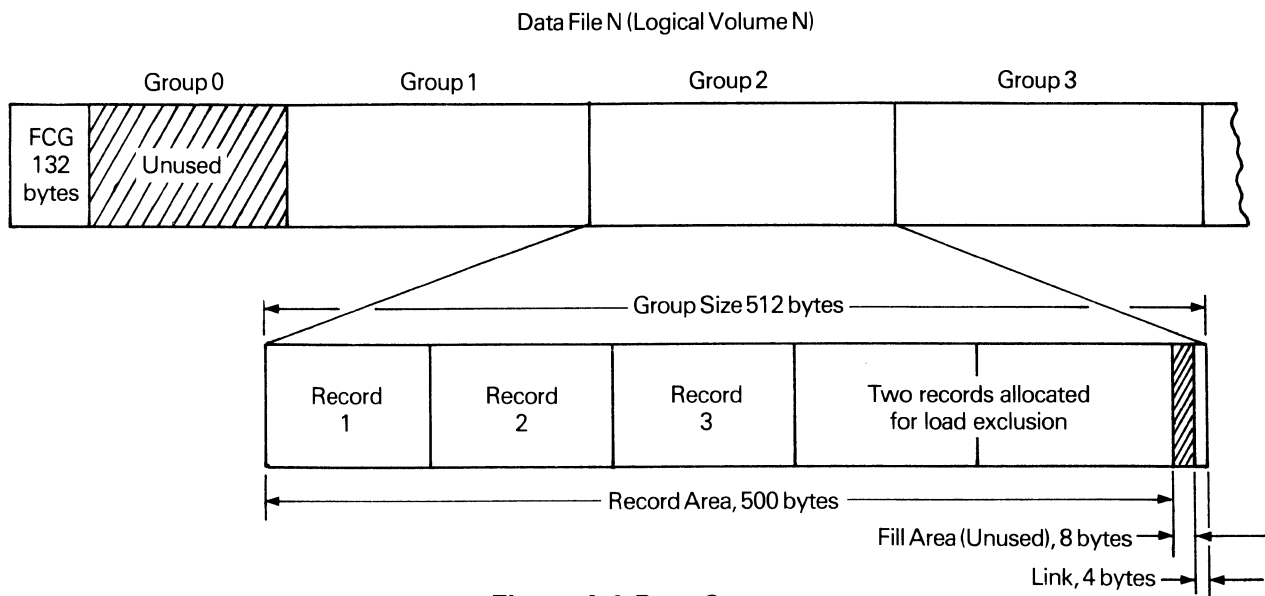


Figure A-2 Data Group

A.3.1.2 Index File — An ISAM file contains one, and only one, index section (file). This is defined as file 0. File 0 is always examined first to determine the status of the entire ISAM file and also to determine the group location of a desired record, or the location of the group in which to store a record.

The entire index file is divided into groups which are the same size as the data file groups. The first group, group 0, as in the data file, is an FCG. The first 132 bytes of this group contain a description of the ISAM file structure, file status, and data file allocations in terms of number of groups. Unlike the data file FCGs, the FCG for the index file is updated as the ISAM file grows and changes. Because CTS-300 ISAM files are designed to be compatible with CTS-500 ISAM files, the FCG contains some information not used by CTS-300. Table A-1 shows the contents of the index file FCG.

Following the FCG group in the index file is the overflow area (if selected) of a size determined by the user. No index entries were set aside for the overflow groups as there were for the data section groups. All records in overflow are accessed via the linking information in each data file group. Once a group is accessed, whether a data file group or an overflow group, the records in that group are accessed sequentially.

The last part of the index section is the index. Since the group size is the same as in the data file; and since the key is smaller than its corresponding record, there are many more index key entries per group than records per group. Each index entry consists of the record key and associated link to its data group or, as you will see, to another index entry. If the keys and their links do not fit exactly into the space available in the group, the unused space is ignored.

To speed access, the index itself can be indexed. The resulting levels of indexing are determined by an algorithm based on the total number of data file groups. In fact, all index entry spaces are preassigned, and links are preconstructed, based on this total number of data file groups. The highest (coarsest) level of indexing appears first in the file and the lowest (finest) level of the index (the largest part) is at the end of the file. The higher index entries point, via index entry links, to lower levels until, at the lowest level, the index entry points to a specific group in a data file. The organization of the index area of the index section is shown in Figure A-3. This is a hypothetical index for an ISAM file of approximately 8000 records with five data records per group and 20 index entries per group. There are seven data files with approximately 1200 records (240 groups) per data file. Keys are numeric, starting with one. The illustration is of a search for a record whose key is 5877. The search is started in the highest (coarsest) level of the index. In this example this level consists of four entries in group 101. The search is for a key of equal or greater value than 5877. The index entry link for index key 5998 points to group 104. The search of this group stops at key 5887, whose associated link points to group 157 in the lowest (finest) section of the index area. The search of the group ends at key 5879 which points, via the link, to data file five, group 257. Group 257 is searched sequentially for the desired record.

Table A-1 File Control Group

No. Bytes	Description	Comment
2	No. of data groups in this file	
2	No. of records per group	
2	Record length in bytes	
2	Key length (1-n)	
2	Key location in record (1-n)	
2	Allow duplicate keys	0 = no, - 1 = add at beginning + 1 = add at end
2	No. retries for locked block	
2	No. levels of indexing	
2	Group No. of first group in primary index	
2	No. index entries per group	
2	No of records per group	
2	Current No. groups in overflow	
2	Maximum No. groups in overflow	
2	Load exclusion factor (0-n)	
2	File protection code	Not used in CTS-300 ISAM (0)
2	Group length	
2	Last file in use (0-7)	
2	File 0 clustersize (2-256)	1
2	File 1 clustersize (2-256)	1
2	File 2 clustersize (2-256)	1
2	File 3 clustersize (2-256)	1
2	File 4 clustersize (2-256)	1
2	File 5 clustersize (2-256)	1
2	File 6 clustersize (2-256)	1
2	File 7 clustersize (2-256)	1
2	Data file 0 allocation	} (No. groups per file, -65535)
2	Data file 1 allocation	
2	Data file 2 allocation	
2	Data file 3 allocation	
2	Data file 4 allocation	
2	Data file 5 allocation	
2	Data file 6 allocation	
2	Data file 7 allocation	
18	[proj,prog]filnam.ex	Not used in CTS-300 ISAM
6	Data file 1 device	} No colon, right space fill
6	Data file 2 device	
6	Data file 3 device	
6	Data file 4 device	
6	Data file 5 device	
6	Data file 6 device	
6	Data file 7 device	
2	Total No. records (low order)	Valid only in index file
2	Total No. records (high order)	Valid only in index file
2	Multikey ISAM	Always 0 for CTS-300 (no multi key)
132	Total bytes	

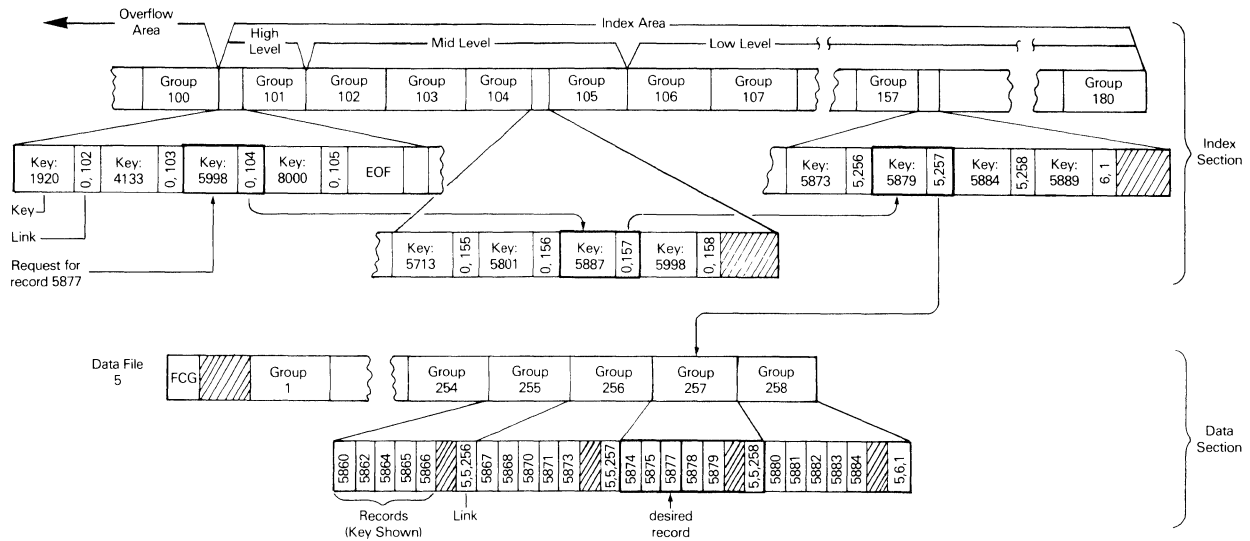


Figure A-3 Index Area

A.3.2 Interrelationships and Tradeoffs

There is no file organization that is best for all applications, but there is one that is better suited to your application. This section discusses choices and decisions for each file component on an individual basis, starting with the key. Physical limits imposed by CTS-300 ISAM are also given. Some of these limits may seem impractically large and should certainly present no problem, but all limits must be observed.

A.3.2.1 Key

Size: 1 to 100 characters (to change limits, see Section A.3.2.3).

The key is an ordinary data field, defined within a DIBOL record statement, which contains alphabetic or numeric information. CTS-300 ISAM can, optionally, accept duplicate keys. That is, more than one record with the same key may be stored. Within the CREATE dialog, you are asked whether the more recent record is to be stored before or after other such records with this key. It is the storage placement that determines the access order. If placed before, it will be accessed first with a READS (sequential read) statement; if after, it will be accessed last after all other records with that key.

If you elect to allow duplicate keys, it is wise to provide sufficient overflow area, because duplicate keys have the potential for unlimited overflow.

A.3.2.2 Record

Size: 2 to 16,383 bytes (to change limit, see Section A.3.2.3).

The most important consideration concerning record size is its relation to group size; therefore, record size cannot be selected without also considering group size. A major design goal for an ISAM file is to fit as many records as possible in a group with little or no wasted space. If the record can be designed so that a multiple will fit the chosen group size, the file will operate at maximum efficiency.

Too many records in a group could cause a performance degradation, because records are always searched sequentially after the group is accessed.

Too few records per group could limit load exclusion availability, and the addition of many records in any such group can result in heavy overflow usage.

A.3.2.3 Changing Record and Key Sizes — The standard record buffer is 1500 characters, and the standard key buffer is 100 characters. However, it is possible to create a file with a record or key length that exceeds these limits. This will require you to change the limits in the UTL2.DBL, RORG3.DBL, and CRET1.DBL subroutines used by the ISMUTL program. Insert the following decimal values as shown below:

```
@@@@ = NEW RECORD LENGTH
%%%% = NEW RECORD LENGTH MINUS 132
+++ = NEW KEY LENGTH
```

Using EDIT:

```
.R EDIT
*EBUTL2.DBL <ESC> <ESC>
*FBUFLN, <ESC> 5J <ESC> 4C@@@@ <ESC> <ESC>
*FKEYLN, <ESC> 5J <ESC> 3C+++ <ESC><ESC>
*EX <ESC><ESC>
```

```
.R EDIT
*EBRORG3.DBL <ESC><ESC>
*FRECBUF <ESC> 1A <ESC> FA <ESC> 4C@@@@ <ESC><ESC>
*6A <ESC> FA <ESC> C%%%% <ESC><ESC>
*5A <ESC> FA <ESC> 4C%%%% <ESC><ESC>
*3A <ESC> FA <ESC> 3C+++ <ESC><ESC>
*EX <ESC><ESC>
```

```
.R EDIT
*EBCRET1.DBL <ESC><ESC>
*FRECBUF <ESC> 1A <ESC> FA <ESC> 4C@@@@ <ESC><ESC>
*6A <ESC> FA <ESC> 4C%%%% <ESC><ESC>
*4A <ESC> FA <ESC> 4C%%%% <ESC><ESC>
*EX <ESC><ESC>
```

After you have modified these routines, you must recompile and relink them as shown below for ISMUTL.SAV (SUD system):

```
.R DICOMP UTL2=FCGFX,RORG1,RORG2
no-error response
```

```
.R DICOMP RORG3=RORG4,STAT,CRET1
no-error response
```

```
.R DICOMP CRET2=CRET3,NUMQ
no-error response
```

```
.LINK/PROMPT/EXE:ISMUTL.SAV UTL2,FCGFX,DATE,DIBOL
RORG1/O:1
RORG2/O:1
RORG3/O:1
RORG4/O:1
STAT/O:1
CRET1/O:1
CRET2,NUMQ/O:1
CRET3/O:1//
```

A.3.2.4 Group

Size: 132 to 16,383 bytes (additionally, the group size cannot exceed 127 records).

The first consideration in determining group size is related to machine I/O. For efficiency, it is best if the ISAM group size is equal to the machine I/O block size. Data is normally read by the hardware in blocks of 512 bytes. Therefore, a group size smaller than the hardware block size would not use part of the data obtained with each read. If the group size is made larger, that is, if a group crosses a block boundary, the cost may be extra I/O for a given record access. Another consideration with groups that cross block boundaries is that they require additional processing by the run-time system.

In time-shared operation, when a record is opened in update mode, the record is locked. Its entire group, and all blocks associated with that group, are also locked and cannot be accessed by another user.

However, I/O byte block size can be changed in whole multiples of 512 bytes with the DIBOL PROC(n) statement. When using the PROC statement you must remember that it affects all files opened by the program. Regardless of group size, however, the link area still requires only four bytes. Therefore, the record space available would be 508, 1020, 1532, etc. bytes for values of n (in PROC(n)) of 1, 2, 3, etc. The use of the OPEN statement to allocate buffer size on an individual basis is a better solution.

The second consideration is to design records to be placed in this group, or to make the group fit a multiple of existing records. If you are free to design the record, this fit can usually be achieved with little waste. On the other hand, if you are creating an ISAM file from an existing sequential file, this cannot be done. In either case, you will likely have to make a decision whether or not to artificially fill any remaining space in the group. This is done at the expense of losing some storage space. However, it avoids the performance degradation caused by mismatching the group and block size. The amount of space wasted by filling has to be balanced on an individual basis by evaluating the increased access efficiency. Ideally, the amount of fill is small, and the decision to fill is then the logical one.

The number of load exclusion records chosen for the group has no effect on the total number of records. The purpose of load exclusion records is to reserve one or more open record spaces within each group. It is selected when you expect to add a random distribution of relatively few records, thus lowering the likelihood of group overflow.

A.3.2.5 Overflow Area

Size: 0 to 16,383 groups

Allocation of overflow area depends on the nature of expected file growth. Overflow is required when new records are likely to be distributed unevenly throughout the file. That is, where the likelihood is great that there will be many records that logically fall under one, or a few, keys. The problem with overflow, as mentioned before, is that access within the overflow area must be accomplished via links and sequential search which involves multiple reads. Within overflow, the advantages of ISAM are lost, except that storage is still sequential. Overflow should be used only as a temporary solution to accommodate a particular class of added records. As the overflow area becomes filled, the need to reorganize the file to increase access efficiency becomes greater. See Section 10.2.5 in Chapter 10. The manipulation of records by ISAM when overflow occurs is illustrated in Section A.3.4.

A.3.2.6 Append Area

Size: 0 to number of records determined by system storage limitations

The append area, since it is indexed, does not have the drawbacks associated with the overflow area. Index entries are created for each append area group as it is filled. Efficient use of an append area is achieved whenever records are added in ascending order by key value. If a record with a high key value is prematurely stored, this could eventually cause records with lower key values to be placed in overflow.

If it is the nature of the application that new data records being added to an existing ISAM file will always, or usually, have a key with a greater value than those records already in the ISAM file, then a large append area would be required. In this case, little or no load exclusion area or overflow area would be necessary since we are not expecting new data records to have to be inserted between existing records.

The append area is assigned in whole groups, even though it is specified in records. Thus, if you had previously chosen four records per group for group size, and specified six records for append, the append area would be eight records (two groups) in size. Also, ISMUTL always appends a minimum of one group. So, even if you requested no append area, in this example there would be space for four records in the append area.

If no input file is specified when creating the ISAM file, the amount becomes the entire data area of the file.

A.3.3 DIBOL Statements

ISAM file access and data manipulation is accomplished with both standard DIBOL statements and special ISAM DIBOL statements. All bookkeeping and updating involved are handled by the run-time system. A complete description of these statements is found in the *DIBOL-83 Language Reference Manual*.

NOTE

All media containing any part of the ISAM file must be on-line for any ISAM DIBOL statement operation.

See Section 5.4.5.1 in Chapter 5 for important information on file sharing.

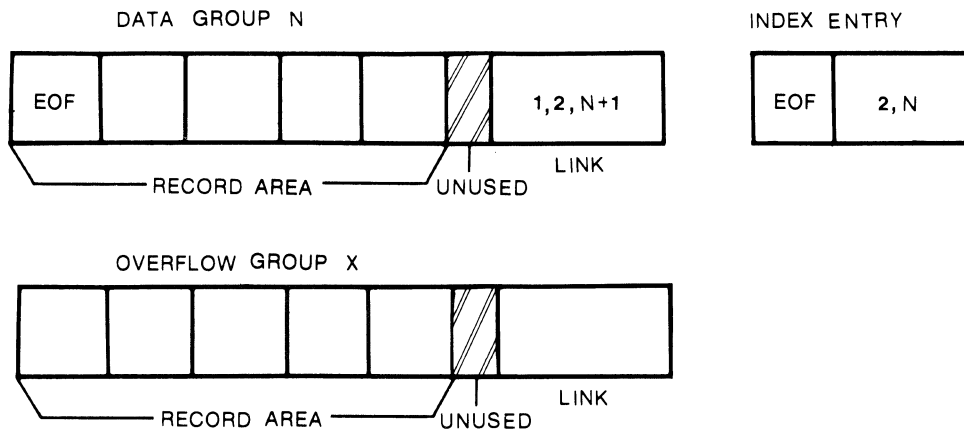
A.3.4 Data Storage

As shown in Chapter 10, ISAM files may be created without any data as input; with a sequential file as input; or with another ISAM file as input. In this section, files will be shown as data is added to illustrate the characteristics of an ISAM file. The first discussion will be of an empty file to which records are added. The second discussion will be of an ISAM file created with a sequential file as input, followed by later addition of records.

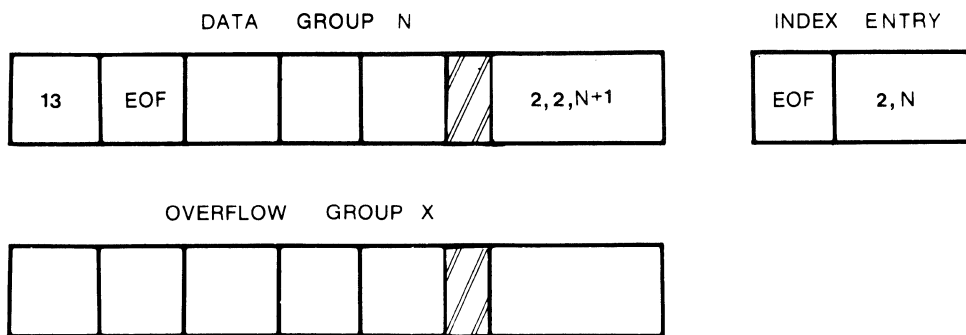
Records Added to an Empty ISAM File

Consider a data file with group number "N". This file was created without any specified input. This is data file two in an ISAM file whose group size is 512 bytes, each of which contains five records of 100 bytes each. Each data group in this file contains (in addition to the five records) 8 bytes of unused space because the fill option was chosen. Changes are shown in this data group as records are added and deleted.

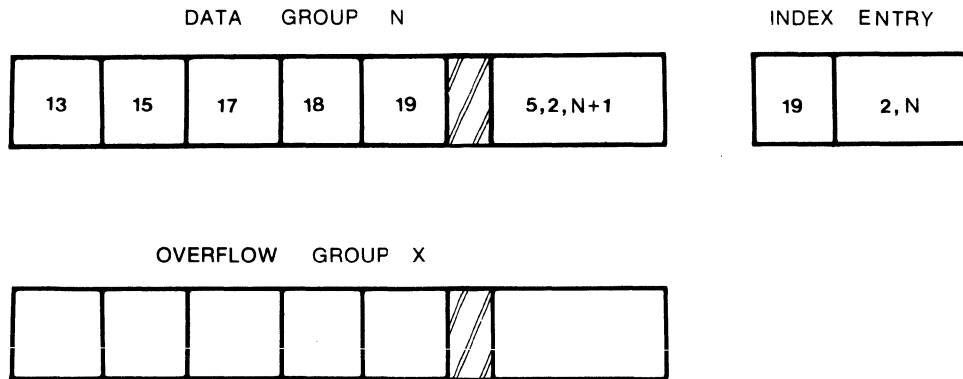
Note that the figures are not to any scale and the information shown consists only of key values, EOF entries, and link information. The link information format is: number of valid records in the group, number of the next data file, number of the next logical group within that file.



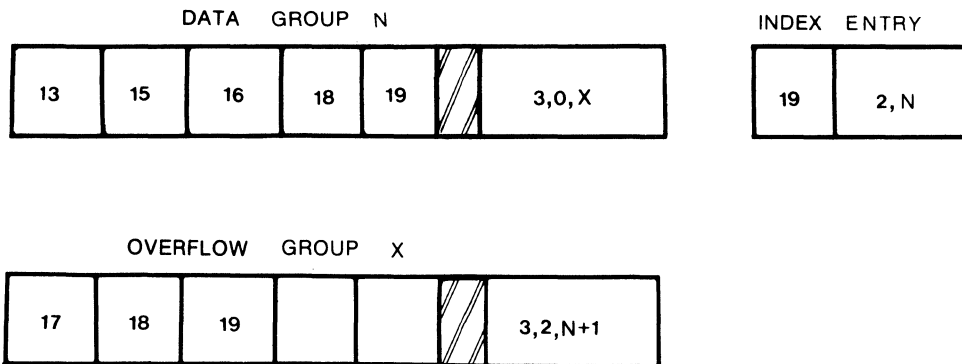
a) Initially the group has no data. The only entry is an EOF in the first record position. The index entry corresponding to the data group also has an EOF as the key entry. The link shows one valid record (the EOF) and points to the next group in the same data file. No search of the data file (or the ISAM file) will proceed beyond this EOF record, since it is the greatest possible key value.



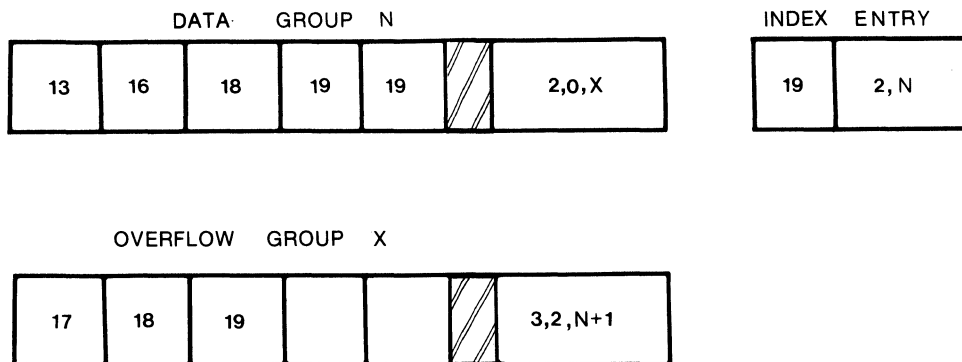
b) A record is added (with the STORE statement) which has a key value of 13. The EOF is now the second record in the group. The link shows two valid records, and the index entry remains unchanged.



c) Records with key values of 15, 17, 18, and 19 are added. The records are stored in ascending order. The EOF record is replaced by the record with the key of 19. The key of 19 is now placed in the index for this group. No record with a key greater than 19 can hereafter be placed in this group number N. The link for the group indicates five valid records.



d) The addition of one more record (key of 16) now exceeds the capacity of the group and results in the split to the overflow area as shown. The index entry for all records in this group is still 19 but now access to records 17, 18, and 19 is achieved via the link to overflow. Data group N contains copies of records 18 and 19, but the link identifies only three valid records in this group. The overflow group link points back to the next logical group (N + 1) in the data file.

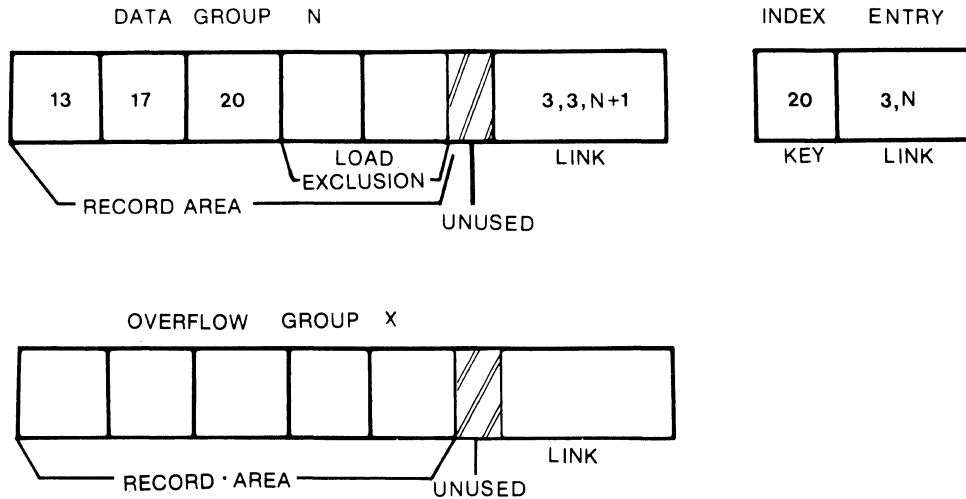


e) If record 15 were now deleted, the group would be as shown. The group N link is unchanged except there are now only the two valid records indicated. Note there are now two copies of record 18 and three of record 19 shown. Only the single entries in the overflow group are accessible.

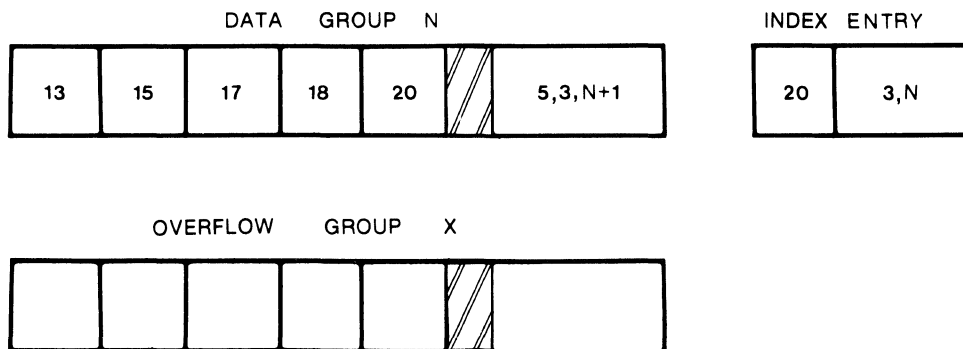
Records Input from a Sequential File during CREATE

When an ISAM file is created with a sequential file specified as the initial input, the user has the option of selecting the load exclusion option. This option is illustrated here. This is data file number three in an ISAM file whose group size, as in the example for an empty file, is 512 bytes, each of which contains five records of 100 bytes each. Each data group in this file contains (in addition to the five records) 8 bytes of unused space because the fill option was chosen. A load exclusion factor of two record spaces has been chosen.

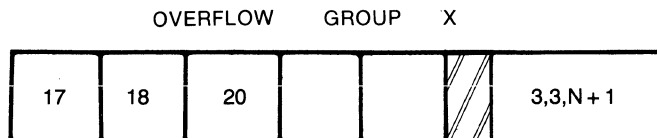
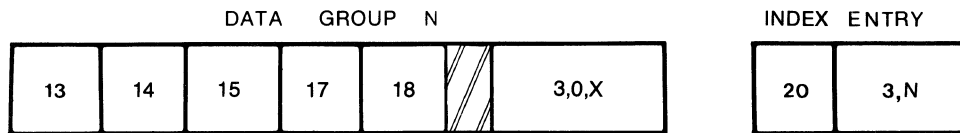
Note that this example, like the empty file example, is not to any scale and the information shown consists only of key values, EOF entries, and link information.



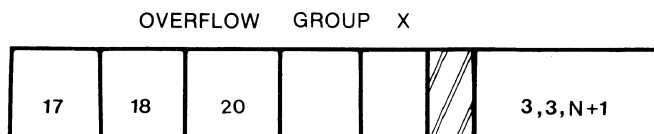
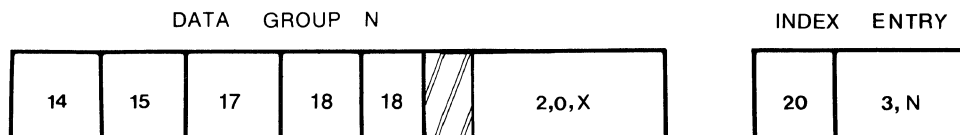
a) When this ISAM file is created, the data from the sequential input file is immediately entered. In this example this group happens to receive records with key values of 13, 17, and 20. Records with greater key values go into the next group because the load exclusion of two records prevents further entries during the creation phase. The key of 20 is placed in the index entry for the group. No record with a key greater than 20 can hereafter be placed in this group N. The link for the group indicates three valid records.



b) At some time after the file is created and all the sequential file data is stored, records are added (one at a time, with the STORE statement) which have key values of 15 and 18. The link now shows 5 valid records, and the index entry remains unchanged.



c) A record with a key value of 14 is now added. The number of records now exceeds the capacity of the group and a split to overflow occurs, as with the example for an empty file.



d) The record with key value 13 is now deleted. The only change is a reduction in the number of valid records in group N and a shift of the other records to the left. This creates a total of three copies of the record with key value 18. Only the one in overflow is accessible.

A.3.5 ISAM File Characteristics

ISAM files and their characteristics have been the topics of discussion throughout Section A.3. However, two characteristics of ISAM files should be particularly noted, because if not kept in mind, they can result in degradation of file access speed.

1. If you have created an empty ISAM file, the file is structured to best accept records in ascending order of key value.
2. If you have created a preloaded ISAM file (that is, an input file was specified during CREATE), the file is structured to best accept new records that will be (a) uniformly scattered throughout the file, or (b) in ascending order - for new record key values that are higher than the highest key value of the old records.

The terminology "structured to best accept" means that a file which has had records entered in the preferred manner can be accessed more readily than a file which was not.

A.3.5.1 Entries in an Empty File — Consider the case where an empty file has records entered in descending order of key value. For the sake of discussion, assume that each group contains space for five records. Assume that there are 100 records to be entered, with key values of 1000, 990, 980, ..., 20, 10.

The first record entered has key value 1000. This record is placed in the first group in the file. The next four records entered are 990, 980, 970, and 960. These records are placed in descending order in the first group in the file. When the group is filled, the key value of 1000 becomes the first entry in the index (low order). In fact, in this case as you will see, this will be the only index entry. The next record entered has the key value 950. ISAM attempts to place this in the first group in the file, but the first group is full. The group then splits with the records containing the higher key values (980, 990, and 1000) going into the overflow area, and the records containing the lower key values (950, 960, and 970) remaining in the first group in the file.

As records with lower key values are added, they will be inserted into the first group until it fills again. A split then occurs, with the higher values going into overflow. This process continues until all the records have been placed in the file. In addition, since the input is never in ascending order, approximately half of the overflow group space will not be used.

The major problem with accessing such a file is that most of the records are in overflow and the overflow area is not accessed randomly, but sequentially. Therefore, if you attempt to access the record with key value 250, you will first access the first group in the file (the group now containing records 10, 20, 30, and 40). This will point to a group in the overflow region. All the records in this group will be sequentially accessed and then point to another group in the overflow region. This continues until record 250 is accessed.

Note that the file is not behaving like an ISAM file at all, but like a sequential file. Further, suppose that while you are accessing record 250, someone else is trying to access record 620. However, accessing record 620 requires passing sequentially through the group containing record 250, and since that group is locked, 620 cannot be accessed. This is true even though the group containing 620 is not itself locked.

Descending order of key entry was chosen as the worst possible case. However, the problems encountered in this case will occur to a lesser extent in any case that is not strictly ascending. If you must enter records into an empty file in non-ascending order, reorganize the file after you are through making your entries. The file will then be structured as if you had made your entries in ascending order.

A.3.5.2 Entries in a Preloaded File — Your ISAM file is structured to most readily accept new records that follow the same pattern of key distribution as the original file. For example, suppose you have an inventory file in which the first few characters of the key field indicate the particular supplier. For the sake of discussion, assume that each group of your file contains space for five records, and that your file has a load exclusion factor of two. This means that when the file was initially populated (from the input file specified during CREATE), each group had three records placed in it, with enough space left open to place two more records later.

If you are supplied with many parts from a single manufacturer, this supplier's records will take up many groups and allow many load exclusion spaces for inclusion of records related to new parts ordered from that supplier. Similarly if another supplier supplies you with only a few parts, only a few groups contain that supplier's records, and therefore only a few load exclusion spaces are available for inclusion of new records. If this second supplier should turn from a minor source to a major source, it is quite likely that the load exclusion space available for this supplier's records will be filled, and new records will be placed in the overflow area. If this happens to any great extent, you will run into the same kinds of problems as discussed in Section A.3.5.1. The solution is to reorganize the file.

Note that you do not have to reorganize the file whenever some records start being placed in the overflow area. It is only when you start to notice degradation in access time that reorganization is recommended.

A.3.5.3 An Overflow Full Error — When the overflow area is full an error is generated. This error means that there is no more room in the file for the record you are trying to store. It does not necessarily mean that there is no room anywhere in the file (or even in the overflow area).

For example, if the overflow area has all its space assigned to overflow groups and you attempt to store a record in a full group - that is, a group in which the load exclusion factor spaces have been filled - you will get an overflow full error. This is true even if there are other groups that still have load exclusion factor spaces available, append area groups available, or even unfilled overflow groups.

A more peculiar situation is the following. Suppose that the entire append area in a file were full, and that the data in the append area were then deleted. Although you have deleted all the data, you have not deleted the entries from the index file. This is an important point because suppose that now you wanted to store a record with a key value greater than that of any record which had been in the append area. In particular, the key value of this record is greater than the key value associated with the last group in the append area. Therefore an attempt is made to place this record in the last group in the append area. Since the group contains the EOF character, it succeeds.

Suppose you keep adding records of higher and higher key value. As long as there is space in this last group, new records will be accepted. However, once that last group is filled, you will get an overflow full error even though most of the append area is empty of data.

A.3.5.4 Determining that an ISAM File Exists and is Not Empty — Some applications require that before opening an ISAM file for update (SU) mode, it must be determined that the file exists and that the file is not empty. The procedure for doing this is usually something like the following:

```
ONERROR NOFILE           ;no-file processing
OPEN (ch,SI,'DEV:filnam.ISM') ;does the file exist as an
CLOSE ch                  ;ISAM file
OFFERROR
OPEN (ch,I,'DEV:filnam.ISM) ;look at FCG
```

The file is first opened in SI mode to determine if such an ISAM file exists and then in I mode so the FCG can be accessed to determine if the file has entries.

There is a problem with this procedure in a time-shared system. The problem is a result of the way the run-time system keeps track of the users of a given file. If two or more users (programs) open the same file as described above, the run-time system will not make the correct table entries identifying the users of the file. This is because the ISAM file was not opened as an ISAM file on the second open and the same device name was used in both cases.

This problem can be avoided if a different device name is used for the open in I mode. The different device name must, however, be for the same logical device.

APPENDIX B

CTS-300 BUILD PROCEDURES

This appendix contains examples of the link structure of all the CTS-300 modules. They are intended to be used as guides for module linking after patches are made.

DBUILD

```
.R LINK
*DBUILD = DBUILD,DIBOL,ODIBOL
* ^ C
```

CTSGEN

```
.R LINK
*CTSGEN = CTSGEN,DIBOL,ODIBOL/C
*GEN1/O:1/C
*GEN2/O:1/C
*GEN3/O:1
* ^ C
```

DICOMP

```
.R LINK
*SJDBL = DIB81/B:2000,DPRSE,EZIORT,NEISLB,TABLES//
*MSGLIB,LOOKUP
*CLIRT/O:1
*XREFRT/O:1
*SCAN/O:1
*INIT11/O:1
*ST11OU,END11/O:1
*LT11OU/O:1
*INITD1/O:1
*LISTTA/O:1
*DPARSE,DSEM/O:2
*PPARSE,PSEM,GEN11,/O:2
*ERRRT,LIST/O:3
*OPENRT/O:3
*ALLOC1,SYMDEF/O:3
*DIREC/O:4
*RARE1,RARE1S/O:4
*RARE2,RARE2S/O:4
//
```

*XMDBL = DIB81/B:2000//
*VIRT,DPRSE,DIRECT,EZIORT,NEISLB
*MSGLIB,LOOKUP
*TABLES *CLIRT/V:1:1
*XREFRT/V:1:1
*SCAN/V:1:1
*INIT11/V:1:1
*END11,ST11OU/V:1:1
*LT11OU/V:1:1
*INITD1/V:1:1
*LISTTA/V:1:1
*DPARSE,DSEM/V:2:2
*PPARSE,PSEM,GEN11,/V:2:2
*RARE1,RARE1S,RARE2,RARE2S
*ERRRT,LIST/V:3:3
*ALLOC1,OPENRT,SYMDEF/V:3:3
*//
* ^ C

DKED

.R LINK
*DKED = DKED,EDLIB,DIBOL,ODIBOL/P:500.//
*COMND/O:1
*COMN2/O:1
*CUTA,CUTB/O:1
*CUTC,TOPB/O:1
*CUTD/O:1
*CUTD0,BEOL/O:1
*DELLN/O:1
*DLCH4,D2CHA/O:1
*D3CHA/O:1
*DQUIT,DSCL1/O:1
*DROPN,SWORD/O:1
*FINDS/O:1
*FIND1/O:1
*HCOMN/O:1
*HELPC/O:1
*HELPD,DEXIT/O:1
*HELPE,CUTD2/O:1
*HWILD/O:1
*PAGE2/O:1
*PASTE/O:1
*REPLC/O:1
*RETRN/O:1
*SECTN,APNDA/O:1
*STRT0/O:1
*STRT1/O:1
*STRT2/O:1
*WPAGE/O:1
*XCASE,LINSP,RESEL,UNDEL/O:1

*CUTC1,CRSTR,UDLCH/O:1
*YANK,ZTARG/O:1
*//
*DKED.TSD/B:100000 = DKED,EDLIB,TDIBOL,ODIBOL//
*COMND/O:1 *COMN2/O:1
*CUTA,CUTB/O:1
*CUTC, TOPB/O:1
*CUTD/O:1
*CUTD0,BEOL/O:1
*DELLN/O:1
*DLCH4,D2CHA/O:1
*D3CHA/O:1
*DQUIT,DSCL1/O:1
*DROPN,SWORD/O:1
*FINDS/O:1
*FIND1/O:1
*HCOMN/O:1
*HELPC/O:1
*HELPD,DEXIT/O:1
*HELPE,CUTD2/O:1
*HWILD/O:1
*PAGE2/O:1
*PASTE/O:1
*REPLC/O:1
*RETRN/O:1
*SECTN,APNDA/O:1
*STRT0/O:1
*STRT1/O:1
*STRT2/O:1
*WPAGE/O:1
*XCASE,LINSP,RESEL,UNDEL/O:1
*CUTC1,CRSTR,UDLCH/O:1
*YANK,ZTARG/O:1
*//
* ^ C

FOCOMP

.R LINK
*FOCOMP = FOC1,MSGLIB/C
*FOC2/O:1/C *FOC3/O:1/C
*FOC4/O:2/C
*FOC5/O:2/C
*FOC6/O:2/C
*FOC7/O:3/C
*FOC8/O:3/C
*FOC9/O:3/C
*FOC10/O:3/C
*FOC11/O:3/C
*FOC12/O:3
* ^ C

ISMUTL

.R LINK
*ISMUTL = UTL2,FCGFX,DIBOL,ODIBOL/C
*RORG1/O:1/C
*RORG2/O:1/C
*RORG3/O:1/C
*RORG4/O:1/C
*STAT/O:1/C
*CRET1/O:1/C
*CRET2,NUMQ/O:1/C
*CRET3/O:1
*ISMUTL.TSD = UTL2,FCGFX,TDIBOL,ODIBOL/B:100000/C
*RORG1/O:1/C
*RORG2/O:1/C
*RORG3/O:1/C
*RORG4/O:1/C
*STAT/O:1/C
*CRET1/O:1/C
*CRET2,NUMQ/O:1/C
*CRET3/O:1
* ^ C

LPTSP1

.R LINK
*LPTSP1 = LPTSP1,MSGLIB/R
* ^ C

LPTSPL

.R LINK
*LPTSPL.TSD = LPTSPL,TDIBOL,ODIBOL/B:100000//
*LPSPL1,LPSPL7,QUP,QUPM/O:1
*LPSPL2/O:1
*LPSPL3/O:1
*LPSPL4/O:1
*LPSPL5/O:1
*LPSPL6/O:1
*LPSPL8/O:1
*LPSPL9/O:1
*//
*LPSAT.TSD = LPSAT,LSATM,TDIBOL,ODIBOL/B:100000
*LQSAT.TSD = LQSAT,LSATM,TDIBOL,ODIBOL/B:10000
*LRSAT.TSD = LRSAT,LSATM,TDIBOL,ODIBOL/B:100000
*LSSAT.TSD = LSSAT,LSATM,TDIBOL,ODIBOL/B:100000
* ^ C

MSGUTL

```
.R LINK
*MSGUTL = MSGUTL,DIBOL,ODIBOL//
*MINID,MINIM/O:1
*MHELP/O:1
*MERRA/O:1
*MERRB/O:1
*MCLSD,MCLSM/O:1//
* ^ C
```

PRINTU

```
.R LINK
*PRINTU = PRINTU,PARSE,DIBOL,ODIBOL
*PRINTU.TSD = PRINTU,PARSE,TDIBOL,ODIBOL/B:100000
* ^ C
```

QUE

```
.R LINK
*QUE.TSD = QUE,TDIBOL,ODIBOL/B:100000//
*QPRS1,QPRS2
*QUP,QUPM
*QAGN *QIN1,QIN1M/O:1
*QIN2/O:1
*QUE1/O:1
*QUE2/O:1
*QUE3/O:1
*QUE4/O:1//
* ^ C
```

REDUCE

```
.R LINK
*REDUCE = REDUCE,MSGLIB
* ^ C
```

RTEXTIT

```
.R LINK
*RTEXTIT.TSD = RTEXTIT,JMAC,TDIBOL,ODIBOL/B:100000
* ^ C
```

SEND/RCV UTILITY

```
.R LINK
*STATUS = SRUTL,BDCON,DIBOL,ODIBOL
* ^ C
```

SORT

```
.R LINK
*SORT = RTIO,SRT11O,SRT11R//
*MSGLIB
*SRT11C/O:1
*SRT11A/O:1
*SRT11D/O:1
*SRT11M/O:1//
*SORT.TSD = SORTR,SRTIO/B:100000//
*SORTC/O:1
*SORTA/O:1
*SORTD/O:1
*SORTM/O:1//
* ^ C
```

SORT/MERGE

```
.R LINK
*SORTG.SAV = SORTG,DIBOL,ODIBOL
*SORTG.TSD = SORTG,TDIBOL,ODIBOL/B:100000
* ^ C
```

STATUS

```
.R LINK
*STATUS.TSD = STATUS,TDIBOL,ODIBOL/B:100000/C
*KMAC/O:1/C
*HDBL/O:1/C
*FDBL/O:1/C
*JDBL,JMAC/O:1/C
*MDBL,MMAC,GMMAC/O:1/C
*SDBL,SMAC/O:1
* ^ C
```

TSD KEYBOARD COMMAND PROGRAMS

```
.R LINK
*COPY.TSD = COPY,TDIBOL,ODIBOL/B:100000
*DEL.TSD = DEL,TDIBOL,ODIBOL/B:100000
*DIR.TSD = DIR,TDIBOL,ODIBOL/B:100000
*RENAME.TSD = RENAME,TDIBOL,ODIBOL/B:100000
*TYPE.TSD = TYPE,TDIBOL,ODIBOL/B:100000
* ^ C
```

UNSUPPORTED SUBROUTINES AND PROGRAMS

.R LINK
*SETVM = SETVM,MAXMM,DIBOL
*CLOCK.TSD = CLOCK,BQLNK,TDIBOL,ODIBOL/B:100000
* ^C

XMTSD CONCURRENT DEVELOPMENT PROGRAMS

.R LINK
*LISTNR = LISTNR *BGMAN.TSD = BGMAN,BQLNK,TDIBOL,ODIBOL/B:100000
*SUBMIT.TSD = SUBMIT,PARSE,TDIBOL,ODIBOL/B:100000
*CANCEL.TSD = CANCEL,PARSE,TDIBOL,ODIBOL/B:10000
*SHOW.TSD = SHOW,TDIBOL,ODIBOL/B:100000
* ^C

APPENDIX C

TIME-SHARED RUN-TIME SYSTEM COMMANDS

These commands are made possible by a feature of time-shared run-time systems that allows any program to be executed by simply specifying its name (the .TSD extension is assumed). In addition, it is possible to have a text string automatically sent to the program when it is specified for running by including the string after the program name. This text string must be separated from the program name by a space or a slash (/). The space is not sent, the slash is. The program name and text string can be up to one line in length. In the following time-shared commands the text string is used to specify options and other files.

All the commands discussed here are terminated with a carriage return.

Error messages associated with any of these commands are documented in the *CTS-300 System Message Manual* under the heading CTSDCL.

The time-sharing system indicates its readiness to receive a command with an asterisk prompt.

C.1 RUNNING A PROGRAM

Two command forms can be issued from any active terminal to load and start execution of a DIBOL program.

Either of the forms shown below to specify a program for execution can be followed by an optional additional character string which could be used to identify a file to be sent (via a SEND statement) to the executed program. This capability is used with the assumption that the executed program will execute a RECV statement to receive the message.

The RUN command has the forms:

*R[un] program [string]

*program [string]

where:

program is the file specification of the program to be executed.

- If not specified, the device DK: is assumed.
- If no extension is specified, the extension .TSD is assumed.
- Certain filenames cannot be used:

R, RU, RUN, A, or ATTACH

string is a string of characters to be sent to the program identified by "program". The receiving program must interpret this string which could be a filespec, two filespecs, or simply a string of text.

Examples:

```
*R BCCT01
```

```
*RU BCCT01
```

```
*RUN BCCT01
```

```
*BCCT01
```

All load and execute program BCCTO1.TSD from device DK:.

```
*R RL1:LIST
```

Loads and executes program LIST.TSD from disk RL1:.

```
*RL1:BCCT01
```

Loads and executes program BCCT01.TSD from disk RL1:.

```
*RUN TEST DL1:FILE.DDF
```

Causes execution of a program called TEST on the system device which then executes a RECV statement to obtain the identification of the input file which in this case is DL1:FILE.DDF.

In all cases program execution continues until one of the following conditions occurs:

- The program issues a STOP or END statement.
- A trappable run-time error occurs and is not trapped by an ONERROR statement.
- A fatal (nontrappable) run-time error occurs.
- A detached program requires the use of a terminal. See the ATTACH command, described in Section C.2.
- The user types a CTRL/C command while the program is attached to the terminal.
- The program is terminated via a kill command from the STATUS program.

C.2 THE ATTACH COMMAND

The ATTACH command connects a program running in the detached state to the terminal from which the command was issued.

The ATTACH command has the form:

```
*A[ttach] filespec
```

where:

filespec is the file specification of the program to be attached to the terminal from which the command is entered.

- If no program name is specified, the command lists all the current detached jobs in the system.
- The default device is DK:.
- The default extension is .TSD.

Examples:

Assume program PGM.TSD is a program that detaches itself when run:

```
.R TSD
TSD VERSION VBnn-nn
*A
NO DETACHED JOBS
*PGM
DETACHING
TSD VERSION VBnn-nn
*A
JOBS RUNNING DETACHED

DK:PGM .TSD
.
.
.
*A PGM.TSD
```

Messages (if any) pending from PGM.TSD would now be displayed.

C.3 THE COPY COMMAND

The COPY command allows the time-shared system user to copy files from one device to another.

The COPY command has the form:

```
*COPY[/option] filespec1 filespec2
```

where:

/option is either the /Q[query] or /NO[query], or /SYS option.

- The option can be placed anywhere in the command line following the COPY command.
- No query is the default. There is no echo.
- The /SYS option allows system (.SYS) files to be copied.

filespec1 is the file specification of the file to be copied.

filespec2 is the file specification of the desired new file.

- If one or both file specifications are omitted, you are prompted for input.
- Asterisk wildcards can be used for file name or extension replacement.
- If no device is specified, the default device DK: is assumed.
- If the extension is omitted, it is the same as specifying a wildcard.
- If the output file name or extension is omitted, a copy is created on the specified device with the input file name and extension.
- Once the copy is completed, a verification is displayed unless the /NOQUERY option was used.

Examples:

```
*COPY/NOQUERY OLDFIL.DDF NEWFIL.*
Files Copied:
DK:OLDFIL.DDF to DK:NEWFIL.DDF
*
```

Copies the file OLDFIL.DDF to a file called NEWFIL.DDF.

```
*COPY DL1:*.DDF DL0:
Files Copied:
DL1:filnam.DDF to DK0:filnam.DDF
(any other files)
*
```

Copies (after the response to the query) all files with a .DDF extension on device DL1: to device DL0:. The names and extensions of the files on DL0: will be the same as those on DL1:.

```
*COPY <CR>
From: XYZ.TXT
To: XYZ1.TXT
Files Copied:
DK:XYZ.TXT to DK:XYZ1.TXT
*
```

Copies file XYZ.TXT to file XYZ1.TXT. Both files are on DK:.

C.4 THE DELETE COMMAND

The DELETE command allows the time-shared system user to delete files.

The DELETE command has the form:

```
*DEL[/option] filespec
```

where:

`/option` is either the `/Q[query]` or `/NO[query]`, or `/SYS` option.

- The option can be placed anywhere in the command line following the DELETE command.
- `/Q` is the default.
- The `/SYS` option allows system (`.SYS`) files to be deleted.

`filespec` is the file specification of the file to be deleted.

- If the file specification is omitted, you are prompted for input.
- Asterisk wildcards can be used for file name or extension replacement.
- If no device is specified, the default device `DK:` is assumed.
- If the extension is omitted, it is the same as specifying a wildcard.
- Once the deletion is completed, the asterisk prompt is displayed.

Examples:

```
*DEL FILE1.DDF
Files Deleted:
DK:FILE1.DDF ? Y
*
```

The Y response deletes file `FILE1.DDF` on device `DK:`.

```
*DEL/NO DL1:*.BAK
*
```

Deletes without query all files with a `.BAK` extension on device `DL1:`.

C.5 THE DIRECTORY COMMAND

The DIRECTORY command produces a listing at the terminal similar to the RT-11 directory command but operates under the time-shared system.

The directory command has the form:

```
*DIR filespec[/option]
```

where:

filespec is the specification for the directory parameters.

- If no device is specified, the default device DK: is assumed.
- If no filespec is specified, a full directory of device DK: is obtained.
- If the device only is specified a full directory of that device is obtained.
- Asterisk wildcards can be used for file name or extension.

option is used to specify one of the options described below:

Option	Description
PRINTER	Attempts to output the directory to the printer if it is free. If not free, an error is returned.
VOLUME NEWFILES BRIEF FREE FULL DIRECTORY BEFORE[:DATE] DATE[:DATE] SINCE[:DATE]	These options operate as described in the RT-11 documentation for the RT-11 DIRECTORY command.

Example:

```
*DIR *.DDF/FULL
```

Would result in a full directory listing of all the files with an extension of .DDF on device DK:.

C.6 THE RENAME COMMAND

The RENAME command allows you to change the name of a file.

The RENAME command has the form:

```
*RENAME[/option] filespec1 filespec2
```

where:

/option is either the /Q[query] or /NO[query], /SYS option.

- The option can be placed anywhere in the command line following the RENAME command.
- /Q is the default.
- The /SYS option allows system (.SYS) files to be renamed.

filespec1 is the original file specification.

filespec2 is the new file specification.

- If one or both file specifications are omitted, you are prompted for input.
- Asterisk wildcards can be used for file name or extension replacement.
- If no device is specified, the default device DK: is assumed.
- Both files have to be on the same device.
- If the extension is omitted, it is the same as specifying a wildcard.
- If the output file name or extension is omitted, a copy is created in the specified device with the input file name and extension.

Examples:

```
*RENAME FIL.DDF FILEA.*  
Files Renamed:  
DK:FIL.DDF to DK:FILEA.DDF  
*
```

Changes the name of file FIL.DDF to FILEA.DDF. The file is on device DK:.

```
*RENAME <CR>  
From: FIL.DDF  
To: FILEA.*  
Files Renamed:  
DK:FIL.DDF to: DK:FILEA.DDF  
*
```

Changes the name as in the first example using the prompt for names.

C.7 THE TYPE COMMAND

The TYPE command allows you to display a file on the terminal.

The TYPE command has the form:

```
*TYPE filespec1...,filespecn
```

where:

filespec1...,filespecn

are up to 9 file specifications of files to be displayed.

- If no file specification is supplied, you are prompted for input.
- Wildcards cannot be used.
- If no device is specified, the default device DK: is assumed.
- If no extension is specified, the default extension .LST is assumed.

Examples:

```
*TYPE LIST1.TXT,DL1:LIST2
```

Displays first file LIST1.TXT located on the default device followed by a display of LIST2.LST that is on device DL1:.

APPENDIX D

CTS-300 OSSL EXTERNAL SUBROUTINES

This appendix describes a group of subroutines supplied with CTS-300. They provide a variety of functions commonly required by DIBOL users. The subroutines can be linked to any DIBOL program using the procedures described in this manual and the *RT-11 System User's Guide*. These subroutines are accessed by the XCALL statement.

FMAC

D.1 FMAC

FUNCTION

FMAC returns the amount of free memory available in either high or low memory. These two values are expressed in bytes. FMAC also returns the size of the calling program and its location in memory, be it high or low.

FORMAT

XCALL FMAC (afield)

where:

afield is an alpha field or record that will contain the returned information.

RULES

- Afield must be at least an 8 character field.
- The information is retrieved in the following format:
 - The first 3 characters indicate free high memory.
 - The 4th and 5th characters indicate free low memory.
 - The 6th and 7th characters indicate the size of the calling program.
 - The 8th character is H if the calling program is in high memory, and L if the calling program is in low memory.
- Each character in the 8-character field represents the following:
 - 1st char = lowest ordered part of the binary number
 - 2nd char = next highest ordered part of the binary number
 - 3rd char = highest ordered part of the binary number
 - 4th char = low-ordered part of the binary number
 - 5th char = high-ordered part of the binary number
 - 6th char = low-ordered part of the binary number
 - 7th char = high-ordered part of the binary number
 - 8th char = calling program is in high memory (H) or low memory (L).

RUN-TIME ERROR CONDITIONS

- 6 NT Incorrect number of arguments
- 31 NT Argument wrong size

EXAMPLES

```

      RECORD MEM
MEMTAB,8A1          Returned information from FMAC
      RECORD
HM1ST, D3
HM2ND, D3
HM3RD, D3
TOTFRH, D6        ;High memory decimal storage
LM1ST, D3
TOTFRL, D6        ;Low memory decimal storage
SIZ1ST, D3
SIZ2ND, D3
PROSIZ, D5        ;Program size decimal storage
      PROC
      .
      .
      .
      XCALL FMAC(MEM)          ;Obtain free memory
      XCALL DECML(MEMTAB(1),HM1STZ) ;High
      XCALL DECML(MEMTAB(2),HM2ND)  ;Memory
      XCALL DECML(MEMTAB(3),HM3RD)  ;Binary value
      TOTFRH = (HR3RD*65536) + (HM2ND*256) + HM1ST ;Calculate decimal value
      XCALL DECML(MEMTAB(4),LM1ST)  ;Low memory
      XCALL DECML(MEMTAB(5),LM2ND)  ;Binary value
      TOTFRL = (LM2ND*256) + LM1ST  ;Calculate decimal value
      XCALL DECML(MEMTAB(6),SIZ1ST) ;Program size
      XCALL DECML(MEMTAB(7),SIZ2ND) ;Binary value
      PROSIZ = (SIZ2ND*256) + SIZ1ST ;Program location (H/L)
      .
      .
      .
      STOP
      END
```


GLINE

D.2 GLINE

FUNCTION

GLINE allows single-user systems to obtain input from an indirect file rather than from the keyboard.

FORMAT

XCALL GLINE([lsize,]lbuffer)

where:

- lsize is an optional argument that indicates the length of the line returned to the caller.
- lbuffer is the line buffer where the input is placed.

RULES

- GLINE is used in place of a READS.
- GLINE should not be used in place of an ACCEPT from the terminal because GLINE requires a RETURN as a terminator before any input is returned to the calling program.
- The system utilities that use GLINE are:

ISAM UTILITY
SORTG
CTSGEN
PRINT UTILITY

RUN-TIME ERROR CONDITIONS

- 6 NT Incorrect number of arguments
- 23 T Line too long

EXAMPLES

None

D.3 R5ASC

FUNCTION

R5ASC changes one RAD50 word to three ASCII characters.

FORMAT

```
XCALL R5ASC(A2,A3)
```

where:

A2 is an alpha field that contains one RAD50 word.

A3 is an alpha field that will contain the ASCII characters.

RULES

- A2 should be a 2 character field
- A3 should be a 3 character field.

RUN-TIME ERROR CONDITIONS

- 6 NT Incorrect number of arguments

EXAMPLES

None

SLICE

D.4 SLICE

FUNCTION

SLICE sets the time allocation count for the calling program.

FORMAT

XCALL SLICE (dfield)

where:

dfield is a decimal field or literal that indicates the job's priority relative to other jobs.

RULES

- Dfield can be any value from 1 to 65,000; the default value is 64.
- Reasonable limits for dfield are between 5 and 200.
- SLICE allows compute-bound programs to run longer under time sharing.
- SLICE is ignored in single-user environment.
- The higher the value set, the longer a program will be allowed to run before being dismissed in favor of another.
- A program is always dismissed when it has an I/O operation pending.

RUN-TIME ERROR CONDITIONS

- 6 NT Incorrect number of arguments

EXAMPLES

```
PROC
.
.
.
XCALL SLICE (20)
```

APPENDIX E

GENERAL NOTES ON I/O

This appendix contains general information on files and file access.

GENERAL

Sequential Access

Sequential access is based on the fact that the run-time system is always positioned or pointing to the next DIBOL record to be read or written.

Random Access

Random access is based on two assumptions: first the data file contains fixed length records, and second, that the record length of the records in the file (excluding the record terminator<CR><LF>) is equal in size to the record area passed in the READ/WRITE statement.

Based on these two assumptions, a calculation is performed to determine where in the file the desired record begins (that is, block number and offset within the block) and how many additional blocks (if any) the record spans.

ISAM Access

ISAM access is based on the fact that records are retrieved sequentially or via "key" designation.

READING AND WRITING A RECORD

Reading

- DIBOL does not transfer to the record area null characters and carriage return characters. These characters are ignored and lost in any transfer.
- Record terminators are the LF (line feed), VT (vertical tab), and FF (form feed) characters. These characters are transferred either, but are recorded as terminators.

NOTE

The ESC (escape) character is not a terminator of DIBOL records.

- The end-of-file character is a CTRL/Z (^Z).
- All characters other than those mentioned above are transferred to the DIBOL record area.
- Different conditions can occur during the transfer:

The record area is filled before a terminator is found.

DIBOL continues to read through the file searching for a terminator. Once found the read is terminated, an error message is generated indicating excessive record size.

The record area is not completely filled when DIBOL detects a record terminator.

DIBOL pads out the record area with spaces.

The record area is equal in size to the record transferred.

A normal transfer occurs.

The end-of-file marker (CTRL/Z) is found.

No padding occurs and DIBOL transfers control to the EOF label designated in the READ statement, goes to the ONERROR label if it is set, or generates a fatal error.

The physical end-of-file is found before a logical end-of-file (CTRL/Z).

This condition prompts the multi-volume question.

Writing

- DIBOL appends a carriage-return line-feed (CRLF) character combination at the end of any record written to a sequential or relative file.

NOTE

This is not true for an ISAM file. Record terminators are not used with ISAM files.

- Depending on the FLAGS external subroutine, a CTRL/Z may or may not be written to a file opened for O mode. If written, it is the end-of-file marker and is placed at the end of the file.

NOTE

Because the CTRL/Z is placed after the last record accessed, it is not necessarily placed at the physical end-of-file.

EXAMPLES

1. Assume the record area of a DIBOL program is equal to the record size of the data file (excluding any terminators) and that the data file has fixed length records. Also assume that the second record slot contains only nulls and no CRLF.

File structure:



A SEQUENTIAL READ:

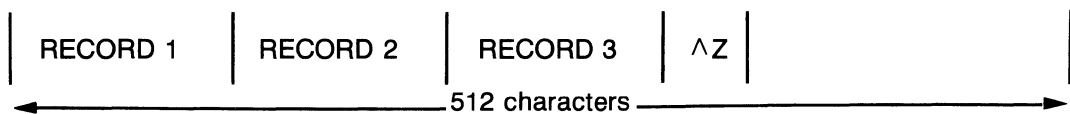
First read obtains DATA 1
 Second read obtains DATA 3
 Third read obtains DATA 4

A RANDOM READ:

Read record 1 obtains DATA 1
 Read record 2 obtains DATA 3
 Read record 3 obtains DATA 3
 Read record 4 obtains DATA 4

2. Assume the record area of a DIBOL program is 600 characters in size. Also assume a file that is one block long (512 characters) which contains three records each with a length of 100 characters and with the fourth record containing an end-of-file marker (CTRL/Z).

File structure:



A SEQUENTIAL READ:

First read obtains RECORD 1
 Second read obtains RECORD 2
 Third read obtains RECORD 3
 Fourth read obtains EOF (end-of-file)

A RANDOM READ:

Read record 1 obtains an illegal record error.

This is because the calculation mentioned above will indicate that two blocks are needed to contain the first record and only one block exists.

GLOSSARY

alpha

A subset of the ASCII character set that contains the 26 letters and 10 digits.

accumulation field

In the PRINTU utility, a field in a record identified as containing data that is to be added (accumulated) from record to record for eventual print out.

append

In ISAM append refers to adding information to the end of an existing file. The end of the file is determined by the record having the highest key.

application program

A program that performs a function (or functions) specific to a particular end-user's needs. For CTS-300 the application program must have at least the root segment written in DIBOL.

argument

A value (variable or constant) supplied with a command, statement, or subroutine call that controls the operation performed.

array

1. A set or list of elements in an ordered pattern.
2. An ordered arrangement of subscripted variables.

ASCII

American Standard Code for Information Interchange. One method of coding alpha characters.

Auto-Create

A method of creating an ISAM file using a file rather than keyboard input to specify the parameters for the ISAM utility program ISMUTL.

Auto job startup

A facility that causes a specified program to be automatically run once the run-time system program is loaded.

background program

A program (always a save file) operating in the background partition of the RT-11 foreground/background architecture. A background program operates automatically, at a low priority, when the higher priority (foreground) program is not using system resources.

backup file

A copy of a file created by the system or by the user for the purpose of protecting file contents in the event the primary file is lost.

base address

1. The address of the first location of a program or data area.
2. An address used as the basis for computing the value of some other (relative) address.

baud

A unit of signaling speed equal to the number of discrete conditions occurring per second. In binary terms a baud rate of one is equal to one bit per second.

bit

A binary digit (0 or 1).

block

1. A group of consecutive words, bytes, characters, or digits considered as a unit.
2. In reference to I/O a block is the smallest system-addressable segment on a mass storage device. The common block size on CTS-300 is 512 bytes.

bottom address

See base address.

breakfield

In the PRINTU utility a breakfield is a field identified as one which determines when accumulated data is printed.

breakpoint

In the DIBOL debugging (DDT) utility a breakpoint is a location at which program operation is suspended to allow operator investigation.

buffer

A storage area used to hold data being passed between devices, between a device and memory, or between memory and the data area of a user application.

bug

An error or malfunction in a machine or program.

byte

A sequence of adjacent binary digits (eight bits in a PDP-11 system).

chain mode

The mode in which a specified program is automatically executed upon the normal termination of another preceding program.

channel

The data path between a processor and an I/O device. Identified in a DIBOL input/output statement by an assigned number between 1 and 15. The number associates that channel with a specified device.

character

A letter, digit, or other symbol used to control or to represent data.

character string

A connected sequence of characters.

checkpoint file

A file created by ISMUTL to record the characteristics and operating parameters associated with an ISMUTL procedure. Its purpose is to enable recovery in the event of a system failure.

command

A word, mnemonic, or character which, by virtue of its syntax in a line of input, causes a computer system to perform a predetermined operation.

command string

A line of input to a computer system that generally includes a command, one or more file specifications, and optional qualifiers.

comments

Programming notes for the user. Comments are optional and are ignored by the compiler.

compile

To produce binary or interpretive code from symbolic instructions written in a high-level source language.

compiler

The program which compiles or translates source level instructions to machine level or interpretive instructions. See compile and DICOMP.

concatenation

The combining in a linear manner of two or more files or strings of characters to produce a longer file or string of characters.

concurrent development

Program development undertaken by simultaneous multiple users.

control character

A character whose occurrence in a particular context initiates, modifies, or stops an operation. In relation to a keyboard, a character (special function key labeled CTRL) that modifies the action of another character. The effect is achieved by holding down the control (CTRL) key and simultaneously pressing the character key.

control file

A file supplied by the user to define the operating parameters of a function (or functions) to be performed by a utility program.

control statement

An element (line or lines) of a control file. Each control statement type has a prescribed syntax.

cross-reference listing

A printed listing that identifies all references in a program to each specific symbol and label in a program. It includes a list of all symbols and labels used in a source program and the statements where they are defined or used.

DCL command

DIGITAL Command Language command. A command which directly causes an operation to take place without further input.

debug

To detect, locate, and remove errors or malfunctions from a program or machine. See DDT.

DEC

Acronym for Digital Equipment Corporation.

default

The value, name, or device that is used if none is specifically requested. Default assignments are a function of the operating or run-time system.

default device

The device assigned by the system if no other device is specified by the user.

default extension

The device extension assigned by the system if no extension is specified by the user.

delimiter

A character that separates, terminates, or organizes elements of a character string, statement, or program.

detached program

A program that operates without association with a terminal. A detached program cannot communicate with the user until it is attached to a terminal.

DDT

DIBOL Debugging Technique. This program is used on CTS-300 for interactive dialog, control, and analysis of DIBOL program operation.

development mode

The mode of operation in which application programs are developed. Contrast this mode with the production mode which utilizes the application program(s) to perform a specific function or functions.

device

A unit or class of hardware such as a magnetic tape drive, disk drive, or other peripheral.

device handler

The software routine that performs I/O and associated tasks for a specific device.

device independence

A characteristic of a program, file, or system that allows it to operate without regard to a particular device type.

device name

A unique name that identifies a device. In CTS-300 the device name size is two characters plus a possible unit identifier number. Example: RK1 is unit 1 of an RK05 disk.

DIBOL

DIGITAL's Business Oriented Language. A high level language used to write business application programs. The source language for CTS-300.

DIBOL Debugging Technique

See DDT.

DIBOL Instruction Set

A special set of hardware instructions supported by the DIBOL language on certain datasystem processors which support the DIS (DIBOL Instruction Set) hardware.

DICOMP

The compiler for the DIBOL language on the CTS-300 Operating System.

direct access

See random access.

directory

1. The area on a disk containing file names and the corresponding physical location of those files on the disk.
2. A place for listing information for reference.

DIS

See DIBOL Instruction Set.

DKED

The DIBOL keypad editor.

duplicate key

In ISAM duplicate key refers to the possibility of having within the ISAM file two or more records with identical key values.

end-of-file

See EOF.

EOF

End-of-file. A character or record that when accessed indicates to the system that the end of the file has been reached.

error log file

A file whose purpose is to store error messages generated by a program running in the detached mode.

external subroutine

A subroutine whose code is physically separate from the calling routine as opposed to a subroutine whose code is a part of the calling routine. See subroutine.

fatal error

See non-trappable.

field

A defined area of a record used for a particular category of data.

file

A collection of related records treated as a unit and usually referenced by a logical name. See indirect file, ISAM file, and sequential file.

file name

The alpha character string used to identify a file and which can be read by both an operating system and a user. In CTS-300 the maximum file name length is six characters. File names are assigned by the user, the application program, or the run-time or operating system.

file name extension

The alpha character string appended to a file name either by an operating system or a user and which can be read by both. In CTS-300 the maximum file name extension length is three characters. The file name extension is sometimes called the file type because it often indicates (by convention) the nature of the file.

file specification

An identifier that uniquely identifies a file. Generally consists of a device name, a file name, and a file name extension.

file structure

The method used to organize records in a file.

file structured device

A device on which data is organized into files. The device usually contains a directory of the files stored.

file type

See file name extension.

fill to end of block

In ISAM fill to end of block refers to the user option of allowing space in a group to be left unused. This is space which is unused after the number of records are chosen for the group. Fill to end of block prevents partial records from existing within a group and prevents records from extending across group (and possibly block) boundaries. Filling creates a file in which records are more quickly accessed at the expense of some wasted space.

fixed-length records

A file in which the records are all of the same length. Direct access is normally possible only with files having fixed-length records.

forced job startup

A feature that allows a program to be started from another program upon execution of a statement within that other program.

foreground program

A program having the highest priority for system resources. A program operating in the foreground partition of the RT-11 foreground/background architecture. A foreground program has priority over a background program.

free blocks

Unused blocks on a file structured device.

global

A symbol, label, or value defined in one program module and referenced by one or more other programs or modules.

group

A group is a user defined division of an ISAM file. A group contains records or index entries. Once specified, group size is constant throughout all parts of an ISAM file.

handler

See device handler.

help frame

Information displayed on a terminal that summarizes the rules and/or parameters for a particular operation or group of operations.

implicit job startup

A feature that operates like forced job startup in that it allows control to be passed to a specified program upon execution of a statement from within another program. In addition, it is possible to request that the specified program be started if not already running.

Indexed Sequential Access Method

See ISAM.

index file

In an ISAM file the section of the total file that contains both the index entries and the overflow area.

indirect file

A file containing a set of commands that could have been entered interactively at a terminal but instead are processed sequentially without operator intervention.

initialize

1. To format a volume in a particular file-structured format in preparation for use by an operating system.
2. To set counters, switches, or addresses to some starting value.

interpreter

In general, a program that translates and executes each source language statement before translating and executing the next statement. In CTS-300 the compiler (DICOMP) does the translation and the run-time system program interprets the translated code for execution.

ISAM

The Indexed Sequential Access Method of file organization, storage, and access. Records are stored and accessed on the basis of the contents of a specified field called a key field. An index is used to determine a position from which a short sequential search is made to access the desired entry. In many applications ISAM provides considerable advantage over random or sequential methods.

ISMUTL

The CTS-300 utility program used to create and maintain ISAM files.

job

A group of data and control statements which does a definable unit of work. In CTS-300 often synonymous with the term program.

key

The field of a stored record that uniquely identifies that record. In random access and ISAM files the key is used as the basis for both storage and access.

key sort

A sort that is done using only record key values and corresponding record addresses.

keyboard monitor

The portion of the resident monitor that provides the interface between the user at a terminal and an operating system.

label

One or more characters used to identify a source language statement or line.

library

A file containing a collection of routines in the form of relocatable object modules (external subroutines). The routines in the library are linked (if required by the program) to a main program object module for use by that program.

link

1. To join two or more object module routines to form one executable program.
2. The information at the end of each ISAM group that identifies the characteristics of that group and the location of the next logical group.

linker

A program that combines two or more relocatable object modules into a single executable module.

literal

An element of a programming language used for the explicit representation of a character or of a character string. In DIBOL-11, a literal is enclosed in quotes to denote that the character or string is to be taken literally and not evaluated.

load

To enter data or programs into memory.

load map

A table produced by a linker that provides information about a load module's characteristics.

load module

A program in a format ready for loading and execution.

location

An address in storage or memory where a unit of data or an instruction is (or can be) stored.

log file

A file used to store information that would normally be output to a terminal.

logical device name

An alpha name assigned by the user to represent a physical device. Limited to three characters in CTS-300.

mass storage device

A device having large storage capacity.

merge

To combine records from two or more similarly ordered strings into another string that is arranged with the same ordination.

message file

In CTS-300 the file containing system and utility error messages, information, dialog, and other text. Programs call messages from this single source rather than generating them individually.

mode

One of the states of possible operation. The state may be the result of a user selection or one of an automatic series of operations. Mode can also refer to a facet of program operation or a file access characteristic (such as: Output, Input, or Update mode).

model

In DKED the model is the specification of the desired character string for a search operation.

monitor

The master control program that observes, supervises, controls, or verifies the operation of a computer system. The collection of routines that controls the operation of user and/or system programs, schedules operations, allocates resources, performs I/O, and other such administrative functions.

multivolume

Having more than one volume. A multivolume file is a file distributed over two or more volumes.

nesting

Placing subroutines, loops, or routines within other subroutines, loops, or routines.

non-fatal

See trappable.

non-trappable

An error that is of such magnitude that should it occur the system is not capable of reliably performing any logical operation.

object program

The code that results after assembling or compiling source code. This code may be linked with other object modules to produce an executable program.

operating system

The collection of programs, including a monitor and system programs, that organizes a processor and peripheral devices into a working unit for the development and execution of application programs and/or run-time systems.

overflow area

An area of an ISAM file located in the index file and of a size determined by the user. ISAM places records in the overflow area that, because of key value, belong in an existing data group but which cannot be placed there because that group is full.

overlay

The technique of temporarily bringing routines into memory from external storage for use during processing. Used when memory requirements exceed memory capacity.

production mode

The mode of operation in which an application program is run to perform a user task.

program

An ordered arrangement of statements that performs a task (or tasks). Usually makes use of subroutines to perform sub tasks. Sometimes called a job.

queue

A list of items waiting to be scheduled or processed according to system or user assigned priorities.

queue manager

A program that schedules and/or processes items in a queue. Queue managers in CTS-300 include BGMAN.TSD and the printer spoolers.

random access

A method of accessing data in which the next data location is not dependent on the location of the previous data accessed. Contrast with sequential access.

record

A collection of related data items treated as a unit. A record contains one or more fields.

remote patching

The ability to make changes in a source file from a remote terminal while the XMTSD Run-Time System is operating.

resident

Pertaining to data or instructions located in memory as opposed to data or instructions located on a storage device.

resident monitor

A group of programs that include terminal service, operating system error handling, system device handler, EMT processor, and system tables.

root segment

The main part of the program. The module that calls the external subroutines. It is associated with external subroutines and library modules by the linking process. For TSD/XMTSD operation the root segment must be written in DIBOL.

routine

A set of instructions arranged in a sequence that causes a computer to perform a desired operation.

RTS

Run-Time System. A group of programs, including an interpreter and facilities for handling input/output, that loads a save program into memory and handles error conditions. Each instruction in the save program is sequentially interpreted to machine code and executed.

RT-11

The basic operating system for CTS-300.

Run-Time System

See RTS.

save program

An executable module which is the result of a previous linking process. A save program is loaded and started with the RUN command.

segment

A physical division on a disk. Each track on a disk is divided into the same number of segments. Data is stored and accessed by a disk handler in relation to segments.

sequential access

A data access method in which records or files are read one after another in the order in which they appear in the file. Fixed-length records are not a requirement for sequential access.

Single-User DIBOL

See SUD.

source program

A program written in a high level language which is later converted to machine language by a compiler. In CTS-300 a source program is written in the DIBOL-11 language.

spooling

The technique by which output to a relatively slow device is placed in queues on a mass storage device until the slower device is able to accept the data.

special keypad

The small keypad to the right of the main keypad on a terminal keyboard. Used extensively with text editors such as DKED.

- statement**
An instruction in a source program.
- string**
A connected sequence of characters.
- subroutine**
A subordinate routine. A separate set of instructions which performs a specified mathematical or logical operation. Usually referenced more than once and from more than one place in a program. A subroutine can be located either within the calling program or external to the program. When located externally, it is called an external subroutine.
- subscript**
A numerical value (or values) that identifies an element or area of an array.
- SUD**
Single-User DIBOL. The SUD Run-Time System supports one user and uses the SJ, F/B, or XM monitor.
- syntax**
The rules governing the structure of a language.
- system configuration**
The combination of hardware and software that make up a usable computer system.
- system device**
The device which the system uses by default to store and retrieve operating system files and other information.
- system program**
A program that performs system-level functions. A program that is a part of the basic operating system. For CTS-300 the operating system is RT-11.
- Time-Shared DIBOL**
See TSD.
- trappable**
An error that does not impair the basic operation of the computer. A trappable error can be detected and investigated by a part of the operating system or run-time system and the results can be displayed or used to make a programmed decision.
- TSD**
Time-Shared DIBOL. The TSD Run-Time System supports multiple users and uses the SJ, F/B, or XM monitor.
- utility**
A program provided to facilitate program development, file maintenance, or other system or support function. Usually supplied with operating systems and run-time systems.
- variable**
A quantity that can assume any one of a set of values.

variable length record

A file in which the records are not of a uniform length. Direct access of such a file is not normally possible.

volume

A single, separately named RT-11 file that can reside on one or more device units of a mass storage medium that can be treated as file-structured data storage (such as a disk cartridge).

wildcard

A special character used in a matching operation in place of one or more characters in a file specification or text string. It's use indicates that, for the character position(s) it occupies, any character will be an acceptable match.

XMTSD

Extended Memory Time-Shared DIBOL. The XMTSD Run-Time System supports multiple users and requires the XM monitor to allow upper memory access.

INDEX

A

Abbreviated keyboard commands, 2-2
Absolute shutdown, 5-29
Access,
 random, 2-5, E-1
 sequential, 2-5, E-1
Accumulation field, 9-2, 9-18
Append area, A-4, A-10
ASSIGN command, 5-13
 use with ISAM file devices, 10-4
 SORT, 12-8
Assignments,
 device, 2-3
 removal of, 2-3
ATTACH command, C-3
Audible alarm, 3-1
Auto-Create (ISAM), 10-19
Auto job startup, specification in
 CTSGEN, 6-18

B

Backup file
 ISAM input, 10-3
 DKED, 3-10
Batch mode, 5-28
BGMAN.TSD, 5-23, 5-25
Breakfield (PRINTU), 9-2, 9-14, 9-15
Breakpoints, see DDT
Build procedures, Appendix B

C

/C (compiler option), 4-3
CANCEL command (XMTSD), 5-27
Carriage return, command terminator, 2-2
CHAIN command (SORT), 12-9
Chain mode startup, time-shared system, 5-16
 RTEXT, 5-29
 STATUS and REORG, 10-16 to 10-18
 time-shared printer spooler, 8-8
Changing record or key length, A-8
Channels,
 specification in CTSGEN, 6-15
 requirement for ISAM files, 10-1, 10-2
Checkpoint files (ISAM), 10-3, 10-13

Cleanup routine (in ISAM CREATE), 10-11
Clearing breakpoints, see DDT Command(s)
 abbreviations, 2-2
 compiler, see compiler, DIBOL
 debugging, see DDT
 DKED, file manipulation, 3-10
 errors from terminal, 2-4
 foreground/background, 5-23
 linker, see program (utility) to be linked
 MSGUTL, 15-7 to 15-9
 syntax, 2-2
 termination, 2-2
 time-shared, 2-1, 5-14 to 5-24
 to allocate system resources, 2-2
 to start a program, 2-3, C-1, C-2
Common cross-reference table
 (compiler), 4-6
Compiler, DIBOL, 1-3, 4-1 to 4-7
 /C option, 4-3
 /D option, 4-3
 /G option, 4-3
 /L option, 4-4
 /O option, 4-4
 /P:value option, 4-4
 /S option, 4-4
 /W option, 4-4
 cross-reference listing,
 see CREF error messages, 4-6
 label table, 4-5
 options (command qualifiers), 4-3 to 4-4
 program listing, 4-4
 running, 4-2
Compiling,
 for DDT, 7-1, 5-6, 5-7, 5-13, 5-20
 with SORTM, 11-18
COMPUTE statement (PRINTU), 9-5
Concurrent development, 5-20 to 5-28
Control file,
 MSGUTL input, 15-9, 15-15
 PRINTU, 9-2, 9-3
 SORT input, 12-5
 SORTGEN, 11-2
Control statements, see PRINTU
COPY command (time-shared), C-4
COUNT command (SORT), 12-10
Creating overlays, 5-14

CREATE, ISAM function, 10-3 to 10-12
 characteristics, 10-3
 dialog, 10-6 to 10-9
 example, 10-10, 10-11
 flowchart, 10-7
 possible CREATE problems, 10-11
 CREF listing (compiler),
 common cross-reference table, 4-6
 external subroutine
 cross-reference table, 4-6
 label cross-reference table, 4-6
 symbol cross-reference table, 4-6
 CTRL/B, LPTSP1.REL, 8-2
 CTRL/C, 2-1, 3-6, 5-5
 CTRL/F, LPTSP1.REL spooler response, 8-2
 CTRL/G, in DDT, 7-3
 CTRL/L, 3-6
 CTRL/O, 2-2
 CTRL/Q, 2-2
 CTRL/S, 2-2
 CTRL/U, 2-2, 3-6
 CTRL/W, 3-3, 3-6
 CTRL/Z, 2-2
 as EOF, 2-5, 12-10, 12-20, E-2
 DKED interpretation, 3-6
 in DDT, 7-3
 CTRL key, 2-1
 CTSGEN, 1-3, 5-1, 6-1, 6-3
 choices, 6-3, 6-4
 dialog, 6-7 to 6-19
 errors, 6-1
 flowchart, 6-6
 hardware/software configuration, 6-17 to 6-19
 preliminary requirements, 6-4 to 6-6
 question types, 6-5
 single-user system, 6-7, 6-8
 terminal specification, 6-8 to 6-19
 time-shared system, 6-10 to 6-20
 CTS-300,
 common file name extensions, 2-4
 data files, 2-5
 device name format, 2-4
 purpose, 1-2
 run-time system errors, see the
 CTS-300 System Message Manual
 system function, 5-6
 utility programs, 1-3, 1-4
 Cursor control (DKED), 3-11
 ADVANCE, 3-13
 BACKUP, 3-13
 BEGIN OF LINE, 3-14
 BOTTOM, 3-14
 EOL, 3-14
 FIND, 3-14
 PAGE, 3-14
 SECTION, 3-14
 TOP, 3-13
 WORD, 3-14
D
 /D (compiler option), 4-3
 Data,
 field (compiler), 4-5
 field dimension (compiler), 4-5
 field names (compiler), 4-5
 field size (compiler), 4-5
 file management (time-shared system)
 5-11, 5-12
 DATE command, 2-3
 DCL-like commands (time-shared
 systems), 5-15,
 Appendix C
 DDT (DIBOL debugging technique), 1-3, 7-1
 breakpoint control, 7-4, 7-5
 command termination, 7-3
 commands, see DDT commands
 compiling for, 7-1
 consequences of improper
 preparation, 7-2
 error messages, 7-3
 linking, 7-2
 preparing for, 7-1 to 7-3
 program execution control, 7-3
 running, 7-3
 specification in CTSGEN, 6-18
 subroutine traceback, 7-8
 variable manipulation, 7-7, 7-8
 DDT commands,
 clearing breakpoints, 7-5
 examining variables, 7-7
 extended variable manipulation, 7-8
 iteration of breakpoints, 7-6
 setting breakpoints, 7-5
 setting variables, 7-7
 single step, 7-4
 start or resume operation, 7-3
 subroutine traceback, 7-8
 DEASSIGN command, 2-3
 Debugging, see DDT
 DECFORM, 1-4
 DECtype, 1-4
 Default extensions, 2-4, 10-4
 Default printers, see LPTSPL.TSD
 DELETE command (time-shared systems), C-5

DETACH command,
 SORT, 12-11
 SORTGEN, 11-4
 time-shared systems, 5-11

Detached program operation, 5-11

Device,
 assignments, 2-3, 10-4, 10-9
 handler changes by SET command, 2-3
 name, 2-4
 sharing, 5-13

DICOMP, 4-1, 4-2, 5-20

DIBOL,
 as a command in SORT, 12-12
 code interpreting, 1-2, 5-6
 compiler, 1-3
 data file, 2-5
 debugging technique, see DDT
 DIGITAL's Business Oriented Language, 1-1
 library, 5-7, 5-14
 object file, 4-1, 4-3
 source file, 4-1, 4-3
 utilities, 1-3, 1-4

DIRECTORY command (time-shared systems), C-6

DKED, 1-3, 3-1
 creating files, 3-5
 cursor control, 3-11
 editing, see Editing (DKED function)
 error messages, 3-3, 3-9, 3-12, 3-19, 3-21
 file backup, 3-10
 file manipulation commands, 3-10
 functions, see cursor control and key functions
 search mode, 3-5, 3-14
 stopping, 3-3
 terminal characteristics, 3-2
 wildcards, 3-5, 3-16, 3-17

Duplicate key, A-8

Dynamic memory allocation, 5-10

E

Editing (DKED function), 3-20
 APPEND, 3-23
 CHNGCASE, 3-23
 CUT, 3-24
 DELCHR, 3-22
 DELIN, 3-22
 DELEOL, 3-22
 OPEN LINE, 3-23
 PASTE, 3-24
 REPLACE, 3-23
 RESET, 3-24
 RUB CHAR OUT, 3-24

SELECT, 3-23
 SUBS, 3-23
 UNDLCHR, 3-22
 UNDELIN, 3-22
 YANK, 3-24

END command,
 SORT, 12-13
 SORTGEN, 11-5

END statement (PRINTU), 9-7

EOF (end of file), 2-5, 12-20

ERMSG.TXT, 15-1

ERROR comand (QUE), 8-12

Error log file, 5-4, 5-28

Error messages, see the *CTS-300 System Message Manual*
 compiler, 4-6
 CTSGEN, 6-1
 DKED, 3-3, 3-9, 3-12, 3-19, 3-21
 SORT, 12-2

ESC (escape sequence use with MSGUTL commands), 15-7 to 15-9

Examining variables, see DDT

EXECUTE command (SORTGEN), 11-6

EXECUTE statement (PRINTU), 9-8

EXIT command (DKED), 3-11

EXIT command (QUE), 8-13

Extended memory monitor (XM), 1-1, 1-2, 5-4

Extensions, default, 2-4, 10-4

External subroutine cross-reference table (compiler), 4-6

F

Fatal error handling, specification in CTSGEN, 6-19

FB monitor, 1-1
 with single-user system, 5-6
 with time-shared system, 5-9

FCG, file control group, A-4, A-5

File,
 conventions, 2-4
 device name, 2-4
 extension, 2-4
 sharing, 5-11
 structure (ERMSG.TXT), 15-2
 types, 2-4

Files,
 checkpoint, 10-24, 10-13
 control (PRINTU), 9-2
 control (SORTGEN), 11-2
 data, 2-5, 10-2, A-3, A-4
 EOF (end of file), 2-5

- index, A-5
- ISAM (CTSGEN), 6-17
- ISAM, see ISAM and ISMUTL
- multivolume ISAM, A-3
- multivolume sequential, 2-5
- object (compiler), 4-1, 4-3
- open at one time (CTSGEN), 6-16
- open for update (CTSGEN), 6-16
- random access, 2-5
- sequential access, 2-5
- source (compiler), 4-1, 4-3
- FIND function (DKED), 3-14
- FLAGS option, 2-5
- FLUSH command (QUE), 8-13
- FMAC external subroutine, D-2
- Forced job startup, 5-15
 - specification in CTSGEN, 6-18
 - time-shared printer spooler, 8-8
- Foreground/background communications (XMTSD), 5-24 to 5-27
 - commands, 5-24 to 5-27

G

- /G (compiler option), 4-3
- GLINE external subroutine, D-4
- Group (ISAM), A-2, A-4, A-5, A-7, A-9
- Group, fill to end of block, A-9

H

- Handler,
 - LPTSP1.REL, 8-3
 - LPTSPL.TSD (XMTSD), 8-8
- HEAD1/HEAD2 statements (PRINTU), 9-9
- HELP command (QUE), 8-14
- HELP frame,
 - DKED, 3-3 to 3-5, 3-17
 - MSGUTL, 15-8
 - STATUS, 13-4, 13-9
 - QUE.TSD, 8-9 to 8-21

I

- IDENT statement (PRINTU), 9-10
- Implicit job startup, 5-16
 - specification in CTSGEN, 6-18
 - time-shared printer spooler, 8-8
- Improperly defined symbols, 4-5
- Index file, A-2, A-5
- Index level, A-7
- Indexed Sequential Access Method,
 - see ISAM IDENT command (SORT), 12-14

- INDEX/LIST statement (PRINTU), 9-11
- Indirect file(s),
 - foreground/background communications, 5-24, 5-27
 - with ISAM auto-create, 10-19
 - with MSGUTL, 15-17
- INPUT command,
 - SORT, 12-15
 - SORTGEN, 11-7
- INPUT statement (PRINTU), 9-3, 9-13
- INTERRUPT command (QUE), 8-16
- ISAM files, 1-4, 2-7, A-1
 - advantages, A-1
 - append area, A-4, A-10
 - basics, A-2
 - characteristics, A-11 to A-16
 - data files, A-2, to A-4
 - data groups, A-2, to A-4, A-9
 - data section, A-2
 - data storage, A-11 to A-16
 - default file extensions, 10-4
 - duplicate keys, A-7
 - file control group, A-4 to A-5
 - file efficiency, A-7, A-9
 - handling added records, A-2
 - index file organization, A-5 to A-7
 - index section (file), A-2, A-5, A-7
 - input files, 10-3
 - key, A-2, A-7
 - links, A-4
 - load exclusion factor, A-2, A-8
 - locked block/record, A-9
 - output files, 10-3, 10-4
 - overflow area, A-2, A-5, A-9
 - overflow full error, A-16
 - record, A-2, A-7
 - record area, A-4
 - reorganization, see REORG
 - sequential search, A-2
 - specification in CTSGEN, 6-17
 - status, see STATUS, ISAM function
 - utility program, see ISMUTL
- ISAM command (SORT), 12-16
- ISMUTL (ISAM utility program), 1-4, 10-1
 - Auto-Create, 10-5, 10-19 to 10-20
 - changing record and key buffer sizes, A-8
 - CREATE function, see CREATE,
 - ISAM function
 - duplicate keys, A-7
 - exiting ISMUTL, 10-16
 - fill to end of group, A-9

REORG function, see REORG, ISAM function
STATUS and REORG in chain mode,
10-16 to 10-18
STATUS function, see STATUS,
ISAM function iteration count, see DDT

K

Key, ISAM,
definition (ISAM), A-2, A-7
duplicate, A-7
Keyboard commands,
MSGUTL, 15-7, 15-9
time-shared system, 2-1 to 2-3, 5-15,
Appendix C
Keyboard monitor (KMON), 2-1
Key functions (DKED), 3-6
control, 3-6
arrow, 3-19
editing, 3-22
KEYS command (SORT), 12-17
Key sort (SORTGEN), 11-14, 11-15
KEYS command (SORTGEN), 11-9

L

/L (compiler option), 4-4
Label cross-reference table (compiler), 4-6
Label table listing (compiler), 4-5
LIST command (QUE), 8-16
Line number (compiler), 4-5
Line printer spooler program, see
LPTSP1.REL and LPTSPL.TSD
Link map, 6-19
Linking, 1-3
for DDT, 7-2
for the SUD run-time system, 5-7, 5-8
for the TSD run-time system, 5-13, 5-14
Links (ISAM), A-4
Listing file, 4-2
LISTNR.SAV (XMTSD), 5-23, 5-24
termination, 5-30
LOAD command, 2-3
Load exclusion factor, A-2
LOCKCC command (SORT), 12-18
Locked blocks/records, 5-12, 5-13
Log file (compiler), 4-3
Logical,
assignments, 2-3
names, 2-3
LPQFIL, 8-5

LPQUE statement,
FORMS argument (LPTSP1.REL), 8-3
FORMS argument (LPTSPL.TSD), 8-8
LPTSP1.REL, 8-1
handler, 8-3
requirements, 8-2
response to error messages, 8-4
run-time dialog, 8-3
shared line printer, 8-2
shared terminal operation, 8-2
starting, 8-3
using, 8-2
LPTSPL.TSD, 8-4
assignment of default printers, 8-7
chain mode startup, 8-8
direct startup, 8-7
file recovery, 8-5
forced job startup, 8-8
handlers, 8-6
implicit job startup, 8-8
relationship to QUE.TSD, 8-4
response to LPQUE statement, 8-8
queue file (LPQFIL), 8-4
starting, 8-6
startup dialog, 8-7
stopping, 8-20
suspension of spooling, 8-5

M

Magnetic tape, using, 5-4
Manipulating variables, see DDT
Memory allocation,
single-user system, 5-8
time-shared system, 5-18
XMTSD background, 5-19
XMTSD foreground, 5-23
Memory requirements,
TSD/XMTSD, 5-9, 5-10
MERGE, see SORT and SORTGEN
MESSAGE command (SORT), 12-19
Message Update Utility, see MSGUTL,
editing (DKED), 3-8
inspection (DKED), 3-8
menu display (MSGUTL), 15-5
no query (REDUCE), 14-3
operating (MSGUTL), 15-6
query (REDUCE), 14-2, 14-3
search (DKED), 3-5, 3-14, 3-19
version number (REDUCE), 14-2, 14-4
Model (DKED search), 3-18
MODIFY command (QUE), 8-18

Monitor error messages, 2-4
Monitors (RT-11), 1-1, 1-2
 see also SJ, FB, XM
MSGUTL utility, 1-4, 15-1
 ABORT without change (Mode 7), 15-7
 ADD a message (Mode 1), 15-6
 control file input, 15-9, 15-17
 DELETE a segment (Mode 3), 15-6
 examples, 15-10 to 15-17
 EXIT with changes (Mode 6), 15-7
 help frame, 15-8
 file structure, 15-2
 indirect command file use, 15-17
 keyboard commands (using ESC
 sequence), 15-7 to 15-9
 LIST a segment on LP: (Mode 5), 15-6
 LIST a segment on TT: (Mode 4), 15-6
 mode menu display, 15-5
 MODIFY a message (Mode 2), 15-6
 message format, 15-3
 segments in message file, 15-2
 terminal input, 15-5
 updating file records, 15-4

N

/N (REDUCE option), 14-3
Name,
 device, 2-4
 file, 2-4
 logical, 2-3
No query mode (REDUCE), 14-3

O

/O (compiler option), 4-4
Object file, 4-1, 4-3
OPEN statement,
 time-shared system use, 5-11 to 5-13
Operating commands (RT-11), 2-1
Operating system (CTS-300), 5-1
OSSL external subroutines, Appendix D
Output,
 file (ISAM), 10-3, 10-4
 file restrictions (ISAM), 10-4
OUTPUT command,
 SORT, 12-20
 SORTGEN, 11-10
OUTPUT statement (PRINTU),
 9-3, 9-7
Overflow area, A-2, A-4, A-10
Overlays, 5-4

P

/P:value (compiler option), 4-4
 \P\ (DKED), 3-5, 3-15
PAD command (SORT), 12-21
PRINT statement (PRINTU), 9-4, 9-18
Printer, see LPTSP1.REL and
 LPTSPL.TSD PRINTU utility, 1-4, 9-1
 /IS option, 9-3
 /SU option, 9-3
 accumulation field, 9-2, 9-18
 break field, 9-2, 9-14
 column separators, 9-18
 COMPUTE statement, 9-5
 control file, 9-2, 9-3, 9-20
 END statement, 9-7
 error messages, 9-2
 EXECUTE statement, 9-8
 field description lines, 9-14, 9-15
 file control records (as a limitation), 9-1, 9-2
 HEAD1/HEAD2 statements, 9-9
 IDENT statement, 9-3, 9-10
 INDEX/LIST statement, 9-11, 9-12
 INPUT statement, 9-3, 9-13 to 9-16
 ISAM input file, 9-11
 OUTPUT statement, 9-3, 9-7
 PRINT statement, 9-4, 9-8
 producing the report program, 9-20, 9-21
 report program, 9-2, 9-20, 9-21
 required control statements, 9-3, 9-4
 running, 9-21
 sorting input prior to using, 9-13
 specification of input file at run time, 9-14
 summary print, 9-3, 9-13
 tag file, 9-3
 tag file as input, 9-11, 9-12
Production mode, 5-28
Program,
 development, 1-3
 development utilities, 1-3
 execution, 2-3, C-1, C-2
 listing (compiler), 4-4
 scheduling, 5-11
Prompt character,
 RT-11, 2-1
 TSD, 5-17
 XMTSD, 5-19, 5-22
PURGE command (QUE), 8-19

Q

Query mode (REDUCE), 14-3
QUE.TSD, 8-4, 8-9
 commands, 8-12 to 8-20

- command format, 8-9
- ERROR command, 8-12
- error listing, 8-13
- EXIT command, 8-13
- FLUSH command, 8-13
- HELP command, 8-14
- help frames, 8-14 to 8-15
- INTERRUPT command, 8-16
- LIST command, 8-16
- MODIFY command, 8-18
- PURGE command, 8-19
- options, 8-9
- queing files for print, 8-10
- relationship to LPTSPL.TSD, 8-4
- RELEASE command, 8-19
- RESERVE command, 8-20
- running, 8-9
- STOP command, 8-20
- TALK command, 8-20
- Queue file (LPQFIL), 8-5
- Queue manager (BGMAN.TSD), 5-23, 5-24
- QUILL, 1-4
- QUIT command (DKED), 3-11

R

- R5ASC external subroutine, D-5
- Random access, 2-5
- R command, 2-3, C-1
- RDCP, 1-4
- Reading a record, E-1
- RECLen command (SORT), 12-22
- RECORD command (SORTGEN), 11-11
- REDUCE utility, 1-4, 14-1
 - /N option, 14-2
 - /V option, 14-2
 - characteristics, 14-1
 - conventions, 14-2
 - error messages, 14-1
 - examples, 14-3, 14-4
 - no query, 14-3
 - query mode, 14-2, 14-3
 - running, 14-2
 - version number, 14-2, 14-4
 - wildcards, 14-4
- RELEASE command, 8-19
- Remote patching (XMTSD), 5-28
- RENAME command (time-shared systems), C-7
- REORG, ISAM function, 10-13 to 10-16
 - characteristics, 10-13
 - dialog, 10-14
 - example, 10-15

- in chain mode, 10-16, 10-17
- possible REORG problems, 10-15
- requirements, 10-13
- RESERVE command (QUE), 8-20
- RT-11 SYSGEN, see SYSGEN
- RTEXTIT,
 - absolute shutdown, 5-30
 - for XMTSD in the foreground, 5-30
 - running, 5-29
- RUBOUT, 2-2
- RUN command, 2-3
 - time-shared systems, C-1
- Run-time system, 1-1

S

- /S (compiler option), 4-4
- SEND/RECV for SUD, 5-7
- Sequential access, 2-5, E-1
- SET command, 2-3
- SETVM (virtual memory handler), 5-20, 5-21
- Single step, see DDT
- Single-User DIBOL (SUD), 1-2
 - linking for DDT, 5-7, 5-8
 - memory allocation, 5-8
 - preparing programs, 5-7
 - running, 5-8
 - SEND/RECV, 5-7
 - system requirements, 5-6
 - with DDT, 5-6 to 5-8
 - with ISAM, 5-6
- Single-user print spooler, see LPTSP1.REL
- Shared line printer operation (LPTSP1.REL), 8-2
- Shared terminal operation (LPTSP1.REL), 8-2
- SHOW command (XMTSD), 5-26
- SJ monitor, 1-1
 - with single-user systems, 5-6
 - with time-shared systems, 5-9
- SLICE external subroutine, D-6
- Sort control record (SORTGEN), 11-20 to 11-22
- SORT utility, 12-1
 - application program level input, 12-3, 12-6
 - ASSIGN command, 12-8
 - CHAIN command, 12-9
 - commands, 12-6, to 12-28
 - control file, 12-5
 - COUNT command, 12-10
 - CTRL/Z as EOF, 12-10, 12-20
 - DIBOL command, 12-12
 - DETACH command, 12-11
 - END command, 12-13
 - error messages, 12-2

- examples, 12-32 to 12-34
- IDENT command, 12-14
- INPUT command, 12-15
- ISAM command, 12-16
- keyboard level input, 12-3, 12-4
- KEYS command, 12-17
- limitations, 12-1
- LOCKCC command, 12-18
- merge specification, 12-15
- MESSAGE command, 12-19
- OUTPUT command, 12-20
- PAD command, 12-21
- RECLen command, 12-22
- rules for using commands, 12-7
- sorting with keys only, 12-24
- SPACE command, 12-23
- specifying a descending order sort, 12-24
- specifying memory available for sort
 - or merge, 12-23
- specifying work files, 12-28
- stopping SORT, 12-6
- summary of commands, 12-29 to 12-31
- tags sort, 12-16, 12-24
- TAGS command, 12-24
- TEMP command, 12-25
- TTNUM command, 12-26
- VERSION command, 12-27
- WORK command, 12-28
- SORTG, 11-1, 11-18
- SORTM, 11-1, 11-18
- SORTGEN, 1-4, 11-1
 - chaining to another program, 11-6, 11-20 to 11-22
 - chain mode, 11-20 to 11-22
 - compiling, 11-8
 - control file, 11-2
 - control key(s), 11-9
 - control commands, 11-3 to 11-16
 - count argument, 11-7
 - DETACH command, 11-4
 - END command, 11-5
 - example, 11-19
 - EXECUTE command, 11-6
 - field definition, 11-11
 - INPUT command, 11-7
 - KEYS command, 11-9
 - linking, 11-20
 - merge specification, 11-3
 - OUTPUT command, 11-10
 - overriding the EXECUTE command, 11-20
 - RECORD command, 11-11
 - required control commands, 11-2, 11-3
 - running, 11-19
 - single-user operation, 11-13
 - program development, 11-18, 11-19
 - SORTG, 11-1, 11-18
 - SORTM, 11-1, 11-18
 - SPACE command, 11-12
 - specifying a descending order sort, 11-9
 - specifying work files, 11-16
 - summary of control file commands, 11-17
 - SU command, 11-13
 - TAGS:INDEX, 11-15
 - TAGS:LIST, 11-14
 - TAGS:SORT, 11-14
 - tags sort with ISAM file as input, 11-15
 - TAGS command, 11-14
 - WORK command, 11-16
- Source code file, 4-1, 4-3
- SPACE command,
 - SORT, 12-23
 - SORTGEN, 11-12
- Special character functions, 2-1
- Spooler, see LPTSP1.REL and SPTSPL.TSD
- STATUS, ISAM function, 10-12
 - example, 10-12
 - in chain mode, 10-16, 10-18
- STATUS utility, 1-4, 13-1
 - available memory, 13-3
 - display update, 13-3
 - error messages, 13-2
 - exiting, 13-8, 13-11
 - features, 13-1
 - job list, 13-4, 13-5
 - kill message, 13-7, 13-11
 - kill job, 13-6
 - list of options, 13-4, 13-9
 - message queue, 13-7, 13-9, 13-10
 - option D, 13-3
 - option F, 13-3
 - option H, 13-4, 13-9
 - option J, 13-4
 - option Jx, 13-5, 13-6
 - option Kx, 13-6
 - option M, 13-7, 13-9
 - option MA, 13-7, 13-10
 - option MKx, 13-8, 13-11
 - option Mx, 13-8, 13-10
 - option T, 13-8
 - option X, 13-9, 13-11
 - single-user systems, 13-9 to 13-11
 - specific job information, 13-5, 13-6
 - time-shared operating parameters, 13-8
 - time-shared systems, 13-2 to 13-9

STOP command (QUE), 8-20
Stopping programs, 5-5, 5-6, 5-29, 5-30
Stored messages, 6-15
SU command (SORTGEN), 11-13
SUBMIT command (XMTSD), 5-25, 5-26
Subroutine traceback, see DDT
SUD, see single-user DIBOL
SUDGEN, see CTSGEN
Symbol cross-reference table listing
 (compiler), 46
Symbol table listing (compiler), 4-5
SYSGEN, 1-3, 5-1, 6-2
System development, 1-3
SYSTEM.BKG (XMTSD), 5-24

T

TAGS command,
 SORT, 12-24
 SORTGEN, 11-14
TALK command (QUE), 8-20
Terminal,
 characteristics with DKED, 3-2
 input (MSGUTL), 15-5
 specification in CTSGEN, 6-8 to 6-14
 shared (LPTSP1.REL), 8-2
TEMP command (SORT), 12-25
Text insertion, 3-22
TIME command, 2-3
Time-shared DIBOL, 1-2, 5-9 to 5-30
 ATTACH command, C-3
 capabilities, 5-9
 commands, 2-1, Appendix C
 common data base, 5-11
 COPY command, C-4
 creating overlays, 5-14
 data file management, 5-11
 DELETE command, C-5
 detached program operation, 5-11
 device sharing, 5-13
 DIRECTORY command, C-6
 dynamic memory allocation, 5-10
 file management, 5-11
 file sharing, 5-11
 keyboard commands, Appendix C
 linking for DDT, 5-14
 linking programs, 5-13, 5-14
 locked blocks/records, 5-12, 5-13
 program preparation, 5-13
 program scheduling, 5-11
 programmed startup, 5-15, 5-16
 RENAME command, C-7

RUN command, C-1
 running programs, C-1, C-2
 scheduling, 5-11
 spooling for printers, 8-4
 stopping programs, 5-29, 5-30
 system requirements, 5-9, 5-10
 TYPE command C-8
 utilizing resources, 5-5
Time sharing, 1-2
Traceback, see DDT
TSD, see time-shared DIBOL
TSD run-time system,
 memory allocation, 5-18
 system requirements, 5-17
 running, 5-17
TTNUM command (SORT), 12-26
TYPE command (timeshared systems), C-8

U

UNLOAD command, 2-3
UNLOCK statement, time-shared system
 use, 5-12
User service routine (USR), 5-5
 specification in CTSGEN, 6-15
Utility programs, 1-3, 1-4

V

/V (REDUCE option), 14-2
Variable manipulation, see DDT
VERSION command (SORT), 12-27
Virtual overlays, 5-20

W

/W (compiler option), 4-4
Wildcards,
 DKED, 3-5, 3-16, 3-17
 REDUCE, 14-4
WORK command,
 SORT, 12-28
 SORTGEN, 11-16
Writing a record, E-2

X

XM monitor, 1-2
 with single-user systems, 5-6
 with time-shared systems, 5-9

XMTSD run-time system

applications as a background program,
5-27, 5-28

applications as a foreground program,
5-27, 5-28

as a background program, 5-18, 5-19

as a foreground program, 5-20 to 5-30

background listener program
(LISTNR.SAV), 5-24

BGMAN.TSD operation, 5-24, 5-25

communication commands, 5-25 to 5-27

foreground queue program

(BGMAN.TSD), 5-24, 5-25

memory allocation as a background
program, 5-19

memory allocation as a foreground
program, 5-23

running in the background, 5-18

running in the foreground, 5-21, 5-22

running LISTNR.SAV, 5-24

system requirements, 5-18, 5-19

termination of LISTNR.SAV, 5-30

termination (RTEXT), 5-29, 5-30

virtual overlays, 5-20

Y

YANK command, 3-24

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or
Country

Please cut along this line.

---Do Not Tear - Fold Here and Tape---

digital



No Postage
Necessary
if Mailed in the
United States

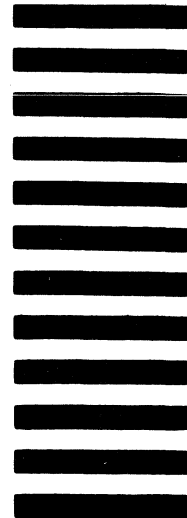
BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Applied Commercial Engineering MK1-2/H32
Continental Boulevard
Merrimack N.H. 03054

ATTN: Documentation Supervisor



---Do Not Tear - Fold Here and Tape---

Cut Along Dotted Line